Universiteit Leiden
The Netherlands

Learning from small samples

Kocaman, V.

Chapter 3

Dealing with Small Data in Machine Learning

In this chapter, we embark on a critical exploration of the myriad approaches to address the unique challenges posed by small sample sizes in machine learning. The genesis of this chapter lies in my quest to find more effective means of training predictive models capable of learning efficiently from limited data. My journey began with an in-depth exploration of batch normalization and gradually evolved into an extensive study of Self-Supervised Learning (SSL), culminating in several papers ([1], [2], [3]) on these subjects.

Realizing a gap that there wasn't a single, consolidated source summarizing the myriad methods and historical perspectives on learning from small samples led to the creation of this chapter; a chapter that started as a necessity for my work but grew into something much larger—a detailed survey and review chapter. This chapter, therefore, is more than just a narrative thread in my dissertation; it is a compilation of a journey, an exploration, and a guide, setting the stage for the deeper and more focused explorations while laying the foundational groundwork, essential for understanding the advanced techniques and specific applications discussed in subsequent chapters.

In the following sections, we navigate through the complexities of overfitting and generalization, and how these are magnified in small data contexts. The chapter then unravels a comprehensive suite of approaches, encompassing techniques like ensemble methods, transfer learning, parameter initialization, loss function reformulation, and regularization techniques. We also discuss innovative methods such as data augmentation, synthetic data generation, and various optimization techniques, all tailored to enhance learning from small datasets. The exploration extends to cutting-edge concepts like physics-informed neural networks, unsupervised, semi-supervised, and self-supervised learning, along with zero-shot, one-shot, and few-shot learning, metalearning, and the harnessing of model uncertainty. Furthermore, we delve into the realms of active learning, self-learning, multi-task learning, symbolic learning, hierarchical learning, and knowledge distillation-based learning, each offering unique insights and solutions to the small data challenge.

Additionally, this chapter addresses specific challenges such as dealing with imbalanced data and anomaly detection as a small data problem, further illustrating the multifaceted nature of these challenges in machine learning. These methods serve as the building blocks for the sophisticated approaches examined in Chapters 4 and 5, specifically focusing on imbalanced data and the impact of self-supervised learning. By the end of this chapter, readers will gain a comprehensive understanding of the fundamental principles and techniques that are pivotal in harnessing the power of small datasets for machine learning.

In sum, this chapter is not an isolated exploration but a vital piece of the puzzle, intricately linked to the advanced techniques and case studies elaborated in Chapters 4 and 5. Therefore, it serves as a narrative thread that weaves through the entire dissertation, setting the stage for a deeper dive into specific applications and methodologies in machine learning when confronted with limited data.

3.1 Introduction

Recent advances in implementing deep learning (DL) techniques have demonstrated that modern algorithms and complex architectures can enable machines to perform tasks with a level of proficiency similar to that of humans. However, it is also evident that a large amount of training data plays a crucial role in the success of

DL models.

It is a well known argument, both theoretically and empirically, that for a given problem, with a large enough dataset, different algorithms tend to perform similarly. It is important to note that this large dataset should contain meaningful information rather than noise, in order for the model to effectively learn from it. The access to large datasets has contributed to the dominance of companies such as Google, Meta, Apple and many more in AI research and product development. Although traditional machine learning (ML) typically requires less data than deep learning, the performance of both traditional ML and DL models is similarly impacted by the availability of large datasets. Figure 3.1 illustrates the performance gap of traditional ML to DL models, with the increasingly growing amounts of data, and proposes a qualitative estimation of the gap between large to small companies. Figure 3.1 basically says that with limited data, traditional algorithms exhibit marginally superior performance over deep learning. As the volume of data increases, deep learning outperforms traditional methods significantly.

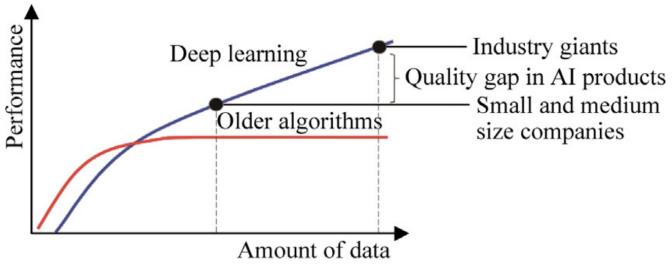

Figure 3.1: A qualitative estimation of performance comparison between traditional ML and deep learning, as well as the performance gap between large to small companies [25]. With limited data, traditional algorithms exhibit marginally superior performance over DL. As the volume of data increases, DL outperforms traditional methods significantly. The red line in this chart indicates the performance of traditional ML algorithms as the amount of data increases. It shows that traditional algorithms start off with a higher performance than DL with small data sizes, but their performance improvement plateaus as the amount of data grows, unlike DL which continues to improve.

Unlike traditional machine learning, human beings are capable of learning about new objects with just a few examples using various approaches, such as relying

on prior experience, comparing known and unlabeled examples, and internalizing semantic descriptions of the object in question. The human learning process, even at early stages, is of particular interest for machine learning, as a ML model initially lacks any prior experience [26]. If we can develop ML models that learn in a similar way, we may be able to approximate human-like learning with small data. Many ML techniques for handling small data are inspired by the human approach, such as data-driven approaches that use prior experience for low-shot problems or learn how to learn tasks, or approaches that replicate human recognition of objects with a few examples through human labeling or semantic description.

The distinction between small and large datasets can be examined further within the context of the Probably Approximately Correct (PAC) model [27]. The PAC model aids in determining the probable characteristics that an algorithm can learn, taking into consideration factors such as sample complexity, time complexity, and space complexity for the algorithm.

PAC basically links the sample size to error rates and to the general success probability, and says that given data X, we expect our algorithm A to output a correct (up to error ϵ) classification hypothesis with probability of at least $1 - \delta$ (δ : failure probability, well-distributed over X). The sample complexity then becomes a function of ϵ and $\delta : m(\epsilon, \delta) = S$ (required examples). This means that for arbitrarily high probability and low error (arbitrarily small δ and ϵ), we can always find a learning algorithm A and a sample size S that achieves that high probability and low error.

The distinction between small and large datasets in the PAC framework is thus related to the model's success probability and error rates in generalizing from the training data to new, unseen data. The distinction between small and large datasets is a critical factor to consider in the design of PAC models, as it affects the generalization performance of the model and the amount of data required to train it. In the PAC framework, a model is considered to have "learned" a concept if it can correctly (up to an acceptable error rate) classify unseen examples with high probability, under certain assumptions about the distribution of the data. Importantly, the PAC framework provides explicit bounds for the sample sizing as a function of the learner's success probability as well as the expected error rate.

Altogether, if the problem is learnable, then satisfying the PAC's bounds on the sample sizing would guarantee the learner's convergence (an event with a certain probability) to a hypothesis that (approximately) classifies the data.

In other words, if the concept is simple and there is a large amount of data available, then a smaller sample size may be sufficient for the model to generalize accurately. However, if the concept is complex and there is a small amount of data, a larger sample size may be required for the model to generalize accurately.

3.2 Handling Small Data

One crucial question that is often overlooked is why we need for ML to address the problem at hand. Consider a scenario where we want to predict the distance a ball will travel when thrown with a velocity (v) at an angle (θ). Using the principles of projectile motion from high-school physics, we can use the equations to precisely determine the pathway of the ball. In this case, the equation can be considered the model or representation for the task, with the variables v , θ, and g (the *gravitational* acceleration) serving as important features. In situations in which exist a few features and a clear understanding of their impact on the task, it is possible to develop a precise analytical model and solve it instantly (mostly by hand).

Now consider a different scenario in which we want to predict a stock price of a company we would like to invest in. In this case, it is difficult to grasp the full range of factors that might influence stock prices. Without a true model, we can use historical stock prices and various other features, other stock prices, and market sentiment, to discover the latent relationships using a ML algorithm. This illustrates a situation where it can be challenging for humans to understand the intricate relationships between a large number of features, but machines can easily capture them by exploring large amounts of data.

In order to build an effective model, it is generally desirable to minimize both bias and variance. This involves creating a model that not only fits the training data well, but also generalizes well to test or validation data. There are various techniques that can be used to achieve this goal wherein training with a larger

dataset is usually the first one.

Imagine a dataset with a distribution resembling a sinusoidal wave. The chart at the upper left corner of Figure 3.2 shows that there are multiple models that are able to fit the data points well. However, many of these models overfit and do not generalize well to the entire dataset.

In DL, overfitting refers to a situation where a model has learned the training data too well, to the point where it memorizes the data and is not able to generalize to new, unseen data. In other words, the model has a high accuracy on the training data, but poor accuracy on the test data. This occurs because the model is too complex and has too many parameters, so it captures the noise and random variations in the training data.

On the other hand, generalization refers to the ability of a model to accurately make predictions on new, unseen data that is drawn from the same distribution as the training data. It is a key aspect of ML, as the goal is to build models that can generalize to new data, rather than just memorizing the training data.

As the amount of data increases, as shown in the charts in Figure 3.2, the number of models that can fit the data decreases. As the number of data points continues to increase, the true distribution of the data is captured more accurately. This example illustrates how a larger dataset can help a model uncover the true underlying relationships.

Another example of a complex task that is challenging for humans but can be easily tackled by ML algorithms is identifying spam emails. Manually developing rules and heuristics to distinguish spam emails can be time-consuming and difficult to maintain. In contrast, ML algorithms can easily learn these relationships from the data and perform the task more accurately and efficiently. This ability to learn relationships from data rather than relying on explicit rules has contributed to the widespread adoption of ML in a variety of fields and industries.

Deep neural networks have millions of parameters to learn and this means we need a lot of iterations before we find the optimal values. If we have small data, running a large number of iterations can result in overfitting. Large dataset helps us avoid

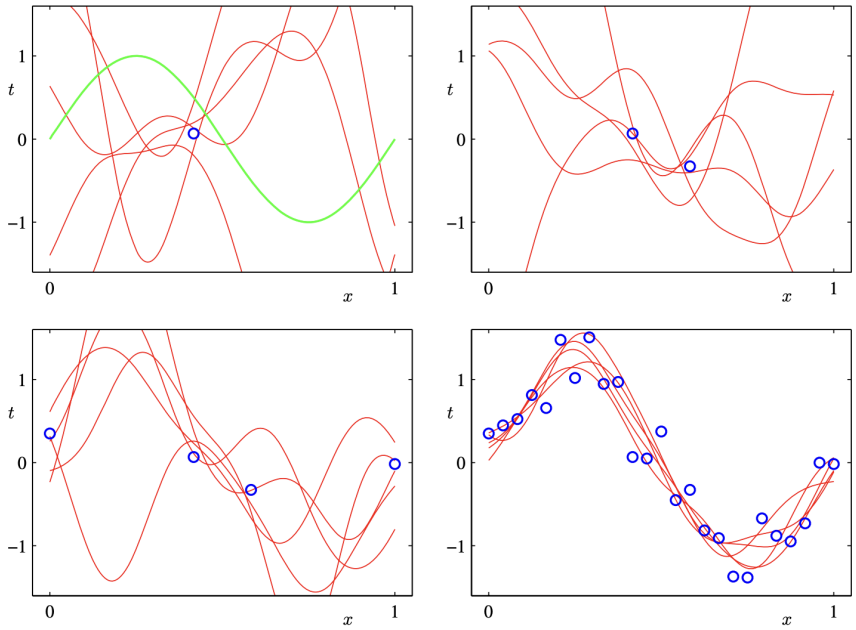

Figure 3.2: Any model can "fit" a single data point (as seen in the upper left corner). As we have more points, less and less models can reasonably explain them altogether. The more data points we have, the better the model capacity to uncover the true underlying relationships; hence less hypotheses we need. As the number of data points continues to increase, the true distribution of the data is captured more accurately (source: [28]).

overfitting and generalizes better as it captures the inherent data distribution more effectively. Due to close relationship of overfitting (caused by the small size of dataset) with the strategies to overcome the lack of enough sample of data points, we will elaborate on the techniques to reduce overfitting (thus enabling generalization) in the following section.

3.2.1 Overfitting and Generalization

The utilization of deep neural networks requires the learning of a vast number of parameters, thus requiring a substantial number of iterations for the optimization course of these parameters. However, in instances where the amount of data that a ML model is trained on is limited, it is more prone to overfitting (especially

when subject to a high number of iterations). This is because there is less feature information to learn from, so the model may end up fitting to the noise or random fluctuations in the training data rather than the underlying pattern. Conversely, utilizing a dataset of ample size can aid in preventing overfitting and yield better generalization by effectively capturing the underlying distribution of the data. As a result, the model may perform well on the training data but poorly on new, unseen data (i.e., when tested/validated).

Reducing overfitting is a crucial step in the training of neural networks as it ensures that the model generalizes well to unseen data. Overfitting can occur when a model is too complex and has seen too little data. To combat this issue, there are several techniques that can be employed from both a dataset perspective and a model perspective.

From a dataset perspective, one effective technique for reducing overfitting is to acquire more labeled data. The more data the model is trained on, the more robust it will be to overfitting. Another technique is to use Data Augmentation and generate synthetic data [29]. This technique allows for the creation of new data points from existing ones, which helps to diversify the training set and reduce overfitting. Additionally, using labeled data from related domains for pretraining via Transfer Learning [30] can be beneficial. By using a model pre-trained on related data, the model will have a better "understanding" of the underlying data distribution, reducing overfitting. Lastly, leveraging unlabeled data for pretraining via Self-Supervised Learning [31] can be a useful technique. By training the model on unlabeled data, the model can learn useful representations of the data, which can be fine-tuned on the labeled data, thus reducing overfitting.

From a model perspective, reducing overfitting can also be achieved by adding regularization to reduce complexity [32]. L2 penalty, weight decay, dropout, and early stopping are all techniques that can be used to reduce model complexity and prevent overfitting [33]. Another technique is to use smaller models, such as the lottery ticket hypothesis [34] and knowledge distillation or to use simpler models that require fewer data points. Lastly, building ensemble models can also be an effective technique for reducing overfitting. Ensemble models [35] combine the predictions of multiple models, which can help to smooth out any overfitting

that may have occurred in individual models (see 3.3.4 for more details regarding ensemble techniques).

In conclusion, reducing overfitting is a crucial step in the training of neural networks, and there are several techniques that can be employed from both a dataset perspective and a model perspective to achieve this goal. We will cover the techniques to reduce overfitting in detail in the following sections.

Recent research [36] has been studying the effect of **pruning** on generalization performance and the findings are intriguing. **Pruning** is a technique to combat overfitting, which is when a model learns the training data too well, including its noise and outliers, to the detriment of its performance on unseen data. By removing some of the weights, pruning simplifies the model, which can help it to generalize better to new data. It has been known for some time that pruning, or producing smaller models, can boost generalization performance. However, other studies have shown that larger, overparameterized models can also improve generalization performance, as seen in double descent and grokking studies. This raises the question of how to reconcile these seemingly contradictory observations.

[36] have found that the reduction of overfitting due to pruning can be partly explained by the improved training process. Pruning involves more extended training periods and a replay of learning rate schedules which can contribute to improved generalization performance. Figure 3.3 illustrates that a pruned model can achieve better generalization performance, not only because of its reduced complexity but also due to the improved training dynamics such as extended training periods and learning rate adjustments. This effect is enhanced in the presence of noisy data, where pruning helps the model to ignore the noise and focus on the underlying patterns, leading to a better generalization on the test data. Additionally, when applied to noisy datasets, the generalization performance improvements due to pruning can be explained by a larger loss on noisy training examples.

Setting larger loss function values on noisy training examples is beneficial because pruned models don't try to fit these noisy examples, which adds a regularizing effect. This is somewhat similar to reducing the width of the layers. This is a new

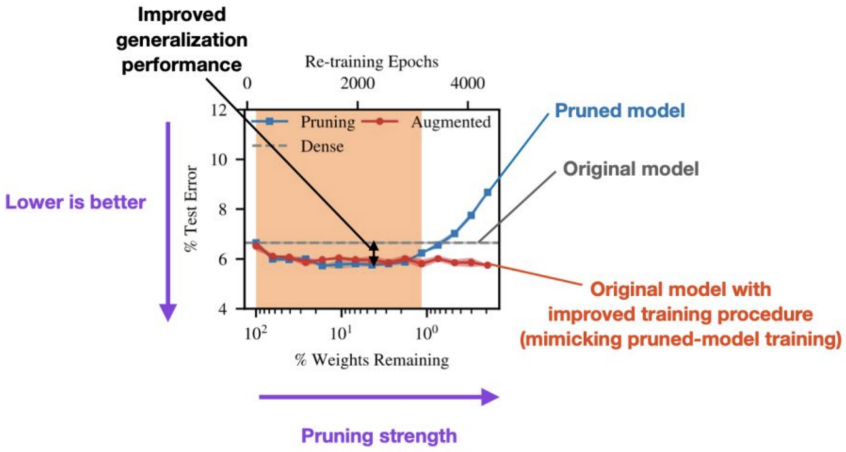

Figure 3.3: Improved training caused by prunning might be responsible for better generalization performance (source: [36]). The blue line represents the pruned model which initially shows improved generalization (lower test error). The retraining and learning rate adjustments during pruning help the model to perform better on test data. The orange line (dense) remains flat across the pruning strength, indicating that the unpruned model's performance does not change with respect to the percentage of weights remaining, as it is not subjected to pruning. The dotted line shows that an improved training procedure can mimic the effects of pruning, leading to a similar generalization performance without actually reducing the model's complexity. This suggests that it's not just the act of pruning that improves performance, but also the associated training techniques. The augmented area suggests that there may be additional methods or data augmentation techniques at play that further improve the generalization performance beyond just pruning.

research area and the findings are very promising, and it will be interesting to see how the research on this subject progresses in the future.

Model averaging is another technique that can be used to mitigate the risk of overfitting and improve generalization performance by combining multiple models. This is achieved by taking the mean or weighted average of the prediction output of each model. While this approach has been shown to be effective in reducing variance and improving generalization, it is important to consider the potential drawbacks, such as the need to maintain and evaluate a large collection of models, which can be computationally expensive and challenging to deploy in a production system.

In terms of generalization, [37] demonstrates that when training neural networks on small synthetically-generated datasets (commonly referred to as algorithmic datasets – a collection of data specifically designed to evaluate the performance of neural networks on tasks that involve symbolic and algorithmic reasoning), unexpected patterns of generalization occur more frequently and drastically compared to datasets obtained from natural sources, even when the performance on the training set is not affected. In [37], as seen in Figure 3.4, the authors show that long after severely overfitting, validation accuracy can sometimes begin to suddenly increase from "chance" level toward perfect generalization, leading to a phenomenon called *grokking* by the authors.

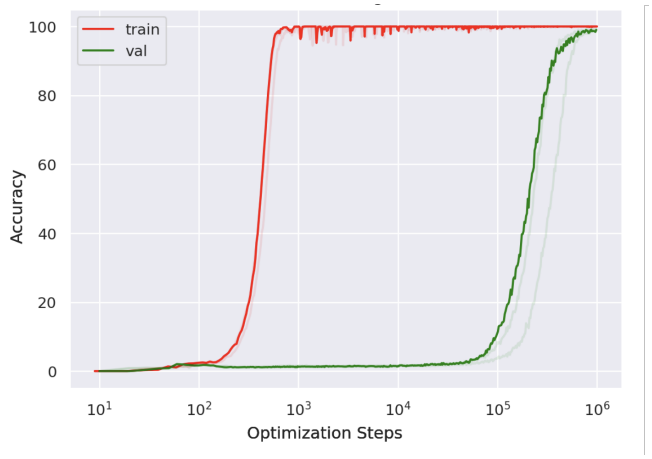

Figure 3.4: A dramatic example of generalization far after overfitting on an algorithmic dataset (a collection of data specifically designed to evaluate the performance of neural networks on tasks that involve symbolic and algorithmic reasoning). The red curves show training accuracy and the green ones show validation accuracy. Training accuracy becomes close to perfect at $< 10^3$ optimization steps, but it takes close to 10^6 steps for validation accuracy to reach that level, and we see very little evidence of any generalization until 10^5 steps (source: [37]).

Another technique to combat overfitting caused by small datasets is regularization, that works by adding a penalty term to the model's cost function that discourages large weights [38]. This helps to constrain the model and make it simpler, which can prevent it from fitting to the noise in the training data (see 3.3.8 for more details on regularization).

We will cover some of these techniques in detail, mostly the ones from a (small) dataset perspective, in the following sections.

3.3 Approaches to tackle small data problems

Small data refers to datasets with a small number of instances, labels, features, or altogether. Dealing with small data poses a challenge in ML as it can lead to overfitting, underfitting, and poor generalization. However, despite the demanding theoretical bounds (e.g., PAC), there are various pragmatic approaches that can be employed to address this challenge and improve the performance of ML models on small data in practice.

Small data can manifest itself in various forms. In some cases, small amount of data that is mostly labelled or unlabelled can be regarded as small data, while in other cases large amount of data that is mostly unlabelled or the number of target classes is partially known (anomaly/ outlier detection) can also be regarded as a small data problem. One of the major challenges of working with small datasets is that the samples may not accurately reflect the underlying distribution of data in the population, leading to difficulties in generalizing the model's performance to new, unseen data.

A small and largely unlabeled dataset is the most infamous version of small data problem wherein both the size of data itself and the labels are not enough to build a reliable model. This situation can pose significant challenges for ML models. The limited amount of data may not provide sufficient information for the model to accurately learn the underlying relationships and patterns in the data. Additionally, the lack of labels can make it difficult for the model to learn from the data, as it lacks the guidance provided by explicit class assignments. This can lead to poor generalization performance on unseen data, as the model may overfit to the limited and potentially noisy examples in the dataset. To mitigate these issues, it may be necessary to carefully select and preprocess the available data, and potentially incorporate additional sources of information, such as domain knowledge.

In summary, there are several approaches that can be employed to deal with small data in machine learning, including regularization, cross-validation, transfer

learning, and data augmentation. Each approach has its own strengths and limitations, and the most appropriate approach will depend on the specific context and goals of the ML task. In the following sections, a few potential approaches that may be useful in this situation will be elaborated.

3.3.1 Data selection and preprocessing

Carefully selecting and preprocessing the available data can help to ensure that the model is able to learn from the most relevant and informative examples. This may involve removing noisy or irrelevant data points, or aggregating or synthesizing additional data to augment the available dataset.

Outliers can have a huge impact on the model and should be identified and removed carefully – overall, it is argued removing the impact of outliers is essential for getting a sensible model with a small dataset. In some cases, outliers could even assist rather than harm the model trained with small datasets [39].

It is hard to avoid overfitting with a small number of observations and a large number of predictors. There are several approaches to feature selection, including analysis of correlation with a target variable, importance analysis, and recursive elimination. It is also worth noting that feature selection will always benefit from domain expertise.

In some cases, the dataset at hand may not exhibit the "small dataset effect", despite its size, if its impact on the learning task is good enough. After all, not all the examples (observations) are equally important and helpful for the model to learn the pattern and generalize over unseen data. Finding the most useful portion of a larger dataset, also known as *coresets*, is another active research area called dataset *prunning*, in which identifying coresets that allow training to approximately the same accuracy as attainable with the original data.

When aiming to achieve a specific target model performance with a limited training data, the question arises: What should be the size of the training subset to achieve a certain performance ? There are a number of different methods (including a prior distribution in Bayesian and frequentist methods that focus on estimation rather than testing) proposed to determine meet this data size requirements [40] [41].

In a similar context, [42] proposed a simple and effective sample size prediction algorithm that conducts weighted fitting of learning curves.

These works attempt to identify examples that provably guarantee a small gap in training error on the full dataset [43]. In this context [44] proposed a scoring method that can be used to identify important and difficult examples early in training, and prune the training dataset with insignificant decline in test accuracy (the examples that can be removed from the training data without hampering accuracy). [44] found out that even at initialization, 50% of the examples can be prunned from the CIFAR-10 dataset without affecting accuracy, while on the more challenging CIFAR-100 dataset, 25% of examples can be prunned resulting in only a 1% drop in accuracy.

3.3.2 Incorporating domain, prior and context knowledge

Human conceptual knowledge has proven difficult for machine systems to replicate in two ways. Firstly, humans are capable of learning new concepts from only a few examples, whereas ML algorithms typically require significantly more examples to perform similarly. This one-shot learning ability allows humans to easily grasp the boundaries of new concepts and make meaningful generalizations, even in the case of children. In contrast, many ML approaches, particularly DL models, require large amounts of data to achieve high levels of performance on tasks such as object and speech recognition. Secondly, human conceptual knowledge tends to be more complex and versatile than that of machines, allowing for the creation of new exemplars, the parsing of objects into parts and relations, and the creation of abstract categories based on existing ones.

One of the key challenges in artificial intelligence and ML is replicating the ability of humans to learn new concepts from just a few examples and to develop abstract, flexible representations. While current ML approaches often require large amounts of data, especially in the case of deep learning, humans are able to learn new concepts and create abstract categories with relatively minimal exposure. Additionally, human learning produces rich representations that can be applied to a wide range of functions, including generating new exemplars and parsing objects into parts and relations, while ML approaches often do not have this capability. A central

question in this field is how to explain these differences in learning between humans and machines, and how to bring the two approaches closer together, particularly given the trade-off between the complexity of a model and the amount of data required for good generalization.

Given this context, Artificial General Intelligence (AGI) may be seen as a potential solution to bridge the gap between human and ML by finding a balance between model complexity and data requirement. AGI [45] refers to a field of AI research that aims to develop algorithms that can perform a wide range of cognitive tasks that are typically performed by humans, including perception, reasoning, learning, and problem solving. The goal of AGI is to create machines that can learn and generalize to new situations, similar to how humans can. The trade-off between the complexity of a model and the amount of data required for good generalization is one of the challenges in achieving AGI. In order to bring the two approaches closer together, researchers are exploring various methods, including developing more sophisticated algorithms, increasing the amount of data available, and improving the interpretability of AI models. AGI has the potential to play a significant role in bridging this gap by enabling algorithms to learn and generalize more like humans.

If there is domain knowledge available that could be used to inform the model's learning process, it may be beneficial to incorporate this information into the model's training. For example, this could involve providing the model with additional constraints or regularization terms to help guide its learning. One way to address the issue of data scarcity is to utilize domain knowledge to restrict the inputs to the model, thereby reducing dimensionality.

Another naive approach would be using rule-based learning, wherein we define rules to arrive conclusions rather than training a model on historical data points. It is also one of the most practical approach to inject weak supervision to the learning process, especially while labelling additional data points (e.g. domain experts write labeling functions that express arbitrary heuristics, which can have unknown accuracies and correlations [46]). Moreover, in order to alleviate the labor-intensive process of manually collecting ground truth labels for ML applications, aggregating multiple sources of weak supervision that is powered by rule based learning reducing the data-labeling bottleneck [47].

Creating rules for labeling data that accurately reflect the characteristics of the data being analyzed usually involves several steps. We start with generating a hypothesis about what the rules should be based on the characteristics of the data (e.g., if the patient's fever is above x and oxygen level is below y, the patient is diagnosed as z). Next, we observe the data to validate the hypothesis and ensure that it accurately reflects the characteristics of the data (using the small sample of observation to validate the rules). Then, we start with simple rules based on the observations made during this process. Finally, we improve the rules over time as more data becomes available and the rules are refined.

The same approach can also be used to reduce false negatives (e.g., choose positive labels when two models disagree) in scenarios such as predicting whether a patient has cancer (it is better to make a mistake telling patients that they have cancer than to fail to detect cancer). Similarly, in order to reduce false positives, choosing negative labels when two models disagree can also be used.

Incorporating domain knowledge through rule-based models have certain limitations. One such limitation is the inability to generalize to unseen data, which can make it difficult to apply these models to new situations. Additionally, it can be challenging to create rules for complex data, and there is no feedback loop to continuously improve the model. Therefore, combining a rule-based model with a ML model may be a more effective approach for data labeling, as it allows for the incorporation of domain expertise while also leveraging the ability of ML models to improve and scale with new data. In order to balance the trade-off between false negatives and false positives, we can decide which type of error to prioritize when combining these two models as briefly mentioned above.

Similarly to context injection, a method of incorporating metadata or contextual information into a ML model in order to improve its accuracy on a specific problem, can also be applied. This approach involves augmenting traditional ML models with contextual information that may be available in the data and can be applied to a wide range of problems, and can be particularly effective when metadata or contextual information is available that is relevant to the task being addressed. The incorporation of contextual information can be achieved through various methods including training a model from scratch with context as an input, using end-to-end

approaches that consider context, or augmenting a pre-trained model by combining context and prediction distributions to make a final prediction.

In another approach called Bayesian program learning (BPL) [48], applying human-level concept learning through probabilistic program induction, the authors construct a model that builds concepts from parts by leveraging prior knowledge in the process. This led to human-level performance and outperforms the DL approaches.

Implicit or explicit sources of domain-knowledge, represented either as logical or numeric constraints (often provided with the help of modification to a loss function) can also be used at the model-construction stage by DNNs as explored in detail by [49]. Implicit Composite Kernel (ICK) [50] is another flexible method that can be used to include prior information or known properties (e.g., seasonality) in neural networks.

3.3.3 Picking the right approach

Deep learning approaches often require a large amount of data to be trained, whereas shallower neural networks and traditional ML methods may require less data and can outperform deep neural networks in scenarios with limited data. In machine learning, small data sets often require models with low complexity or high bias to prevent overfitting. In addition to Support Vector Machines (SVM), decision trees and Gaussian processes (GP), Bayesian methods such as the naive Bayes classifier and ridge regression, are often the best choice for very small data sets, although the results may depend on the choice of prior knowledge. SVM has a particularly good generalization performance in the case of high-dimensional data and a small set of training patterns, without using non-kernel methods [51].

The Naive Bayes algorithm, which is based on the assumption of conditional independence between features given the class variable, is a simple classifier that performs well on small data sets. Linear models and decision trees (or ensembles of them, also known as random forests), which also have a relatively small number of parameters, can also be effective on small data sets. In general, models that have fewer parameters or strong prior assumptions are more suitable for small data sets.

Bayesian neural networks (BNNs, Bayesian NNs) also offer a probabilistic inter-

pretation of DL models by inferring distributions over the models' weights. The model offers robustness to over-fitting, uncertainty estimates, and can easily learn from small datasets [52]. As it is the case with Naive Bayes algorithm, Bayesian algorithms naturally incorporate a form of regularization (the prior), hence less prone to over-fitting the small dataset.

Another study, *Modern Neural Networks Generalize on Small Data Sets* [53], utilizes a linear program to decompose fitted neural networks into ensembles of low-bias sub-networks, which are found to be relatively uncorrelated. This process, similar to a random forest, leads to an internal regularization effect that contributes to the surprising resistance of neural networks to overfitting, even when trained on a small number of examples. In experiments that contain a much smaller number of training examples, deep neural nets are shown to achieve superior accuracy without overfitting. This study argues that building your networks deep enough would let you take advantage of this ensemble effect on small datasets.

In another study [54], there has been some success using stacked autoencoders to pre-train a network with more optimal starting weights, which helps avoiding local optima and other pitfalls of a bad initialization. The same study also claims that fully connected DNN that consists of 3 or more hidden layers shows its advantage over shallow neural network and support vector machine by achieving higher prediction accuracy and better generalization performance.

[55] empirically shows that, by issuing set-valued classifications, *Naive Credal Classifier-2* (NCC2) is able to isolate and properly deal with instances that are hard to classify in low data regimes (on which naive Bayes' accuracy considerably drops), and to perform as well as naive Bayes on the other instances.

In a nutshell, altering the architecture of the model can potentially improve its ability to learn from a limited amount of data.

3.3.4 Ensemble methods

Combining the predictions of multiple models, either by averaging or through a voting process, can often improve the overall performance of the model. This can be particularly useful in the case of small and largely unlabeled datasets, where

individual models may be prone to overfitting or underfitting.

Ensemble techniques [35], which involve aggregating the predictions of multiple models, can often result in improved accuracy and reduced variance compared to individual models. This can be achieved through various methods such as weighting the predictions of different models or using different values of hyperparameters for the same model. Such techniques can be particularly useful in situations where data is limited, as they can help mitigate overfitting and increase generalizability. It is also advisable to seek input from domain experts when implementing ensemble methods, as they may be able to provide valuable insights on the appropriate weighting of different models or the selection of suitable hyperparameters.

In ensemble learning, multiple models are combined to make a final prediction. Two common approaches in ensemble learning are bagging and boosting. Bagging (bootstrap aggregation) involves generating random samples of the training dataset with replacement, running a learning algorithm on each sample, and then taking the mean of all predictions. Boosting is an iterative method that adjusts the weight of each observation based on the previous classification. This approach aims to reduce bias error and build strong predictive models.

3.3.5 Transfer learning

If there are similar tasks or datasets for which labeled data is available, it may be possible to leverage this data to improve the performance of the model on the small and largely unlabeled dataset. This can be done through transfer learning, where a pre-trained model is fine-tuned on the new dataset. Transfer learning [30], also known as domain adaptation, is basically the leveraging the knowledge of a neural network learned by training on one task to apply it for another task (Figure 3.5). The learned weights of a model that was pre-trained on one dataset are used to 'bootstrap' training of early or all but the final layer of a modified version of the model applied to a different dataset.

In transfer learning, the knowledge gained from a source task is utilized to facilitate and accelerate the learning process for a new target task. This technique allows faster training, whereby the model just learns the weights of the last fully connected layers, then applies a low learning rate with finely tuned adjustment to the entire

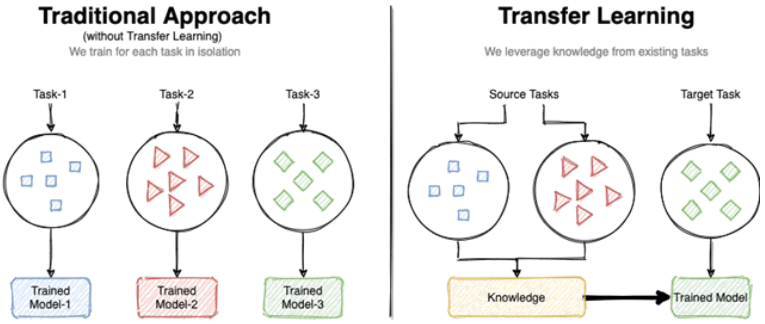

Figure 3.5: In traditional approaches, a separate model is created for each task, as illustrated in the figure on the left where three distinct models are used for three different tasks. In contrast, transfer learning utilizes knowledge acquired from source tasks to enhance the performance of the target task, as depicted in the figure on the right (source: [56]).

model's weights. This is achieved by two main concepts: Freezing and fine-tuning. It is a common strategy to freeze the layers in neural-networks that are supposed to be leveraged from the pre-trained model 'as is', and no weight updates are expected on these layers. Freezing the weights for the initial layers of the network usually gives better results on the smaller target set classes. [57] states that freezing the first two to three layers of features results in a significant performance boost over the baseline score, especially for smaller target set sizes under a thousand instances per class.

In transfer learning, it is important to ensure that the weights trained on the source task are relevant to the target task. If the weights are not relevant, transfer learning may not be effective. It has been observed that the benefit of a pre-trained network greatly decreases as the task the network was trained on diverges from the target task [58]. For example, if the source task involves classifying horses and zebras and the target task involves detecting benign and malign tumors, the weights from the source task may not be useful for the target task. In order to obtain good results, it is important to initialize the network with pre-trained weights that are relevant to the target task.

One of the key challenges in using transfer learning techniques is to ensure that the knowledge acquired from a source task is effectively transferred to a target

task, while avoiding negative transfer between tasks that are not closely related. Removing layers from a pre-trained model may also have negative effects, as it can alter the architecture of the model and potentially result in overfitting. It is therefore important to carefully consider the number of layers to include in the model.

In computer vision, deep neural networks trained on a large-scale image classification dataset such as ImageNet have proven to be excellent feature extractors for a broad range of visual tasks such as image classification and object detection. The sample illustration of transfer learning and details of a typical CNN architecture for image recognition can be seen at Figure 3.6.

Another important observation is that the performance of ImageNet architectures and the power of transferability across different datasets may differ given the type of the network (e.g., VGG-19, ResNet) and the nature of the dataset (see Figure 3.7 for more details).

There are different forms of transfer learning:

Inductive transfer learning refers to the use of knowledge learned from one task to improve the performance of another task with the same structure, but with different data. It is commonly used in real-world settings and is effective at improving the performance of the target task. This type of transfer learning may involve multitask learning, in which the model is trained on multiple related tasks, or self-taught learning, in which the model is trained on a large amount of unlabeled data and then fine-tuned on a small labeled dataset [56].

Inductive transfer learning is particularly useful when the target task has a limited number of labeled samples, as it allows the model to make use of the larger labeled dataset in the source domain (labeled data from both source and target domains are available for training). In order to induce the knowledge from the source domain, it is necessary to have labels available in the target domain. Inductive transfer learning is commonly used in real-world settings and is effective at improving the performance of the target task.

In a recent study, entitled *Simplified Transfer Learning for Chest Radiography*

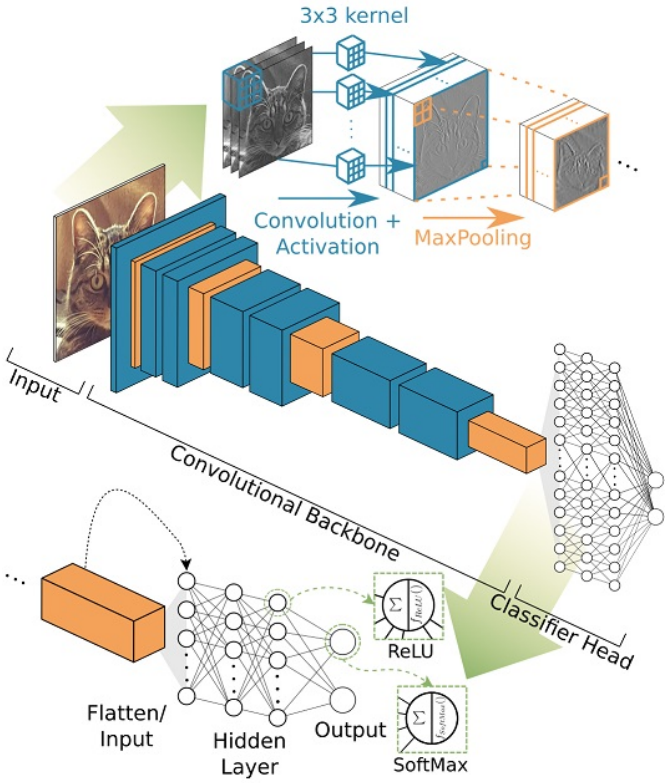

Figure 3.6: Schematic representation of a Convolutional Neural Network (CNN) architecture for image recognition [59]. The process begins with the input image, which undergoes a series of convolutional layers with applied activation functions, followed by max pooling layers to reduce dimensionality while retaining important features. This sequence forms the convolutional backbone, which is crucial for feature extraction. The extracted features are then flattened and fed into a dense hidden layer. The network concludes with a classifier head, consisting of additional dense layers with ReLU activation and a final SoftMax layer for output class probabilities. Each step in the architecture is designed to progressively abstract and condense the image information into a form that the model can use to make accurate classifications.

Models Using Less Data [61], the authors proposed an approach called supervised contrastive (SupCon), and compared with transfer learning from a nonmedical dataset. The models using the pretrained weights reduced label requirements up to 688-fold and improved the area under the receiver operating characteristic

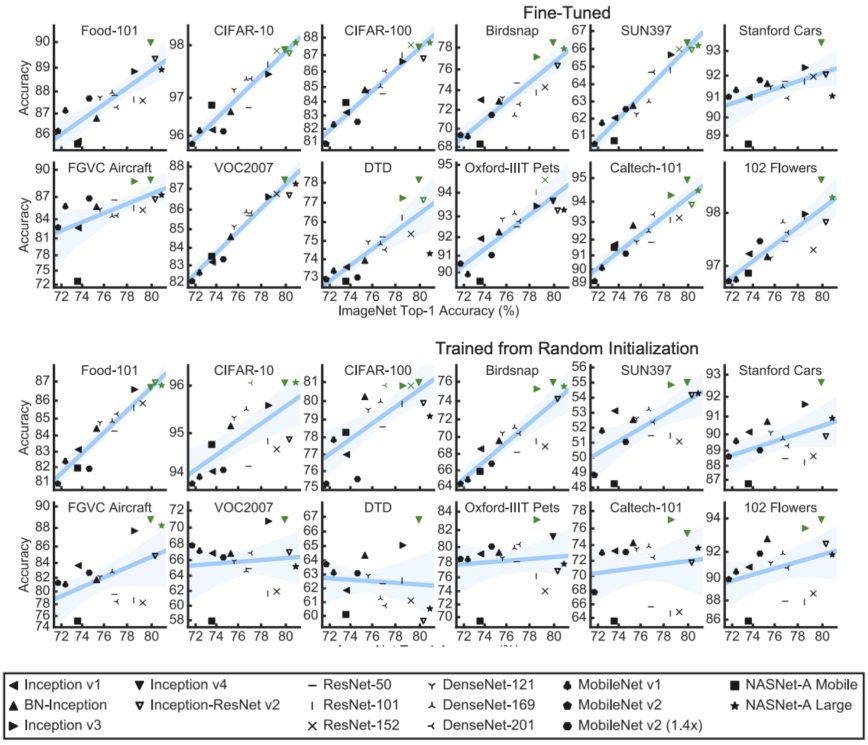

Figure 3.7: The performance of modern CNN architectures on popular datasets differ based on the nature of the datasets and whether the network is initialized from the ImageNet checkpoints (finetuned) or trained from scratch with random initialization (source: [60]).

curve (AUC) at matching dataset sizes. At the extreme low-data regimen, training small nonlinear models by using only 45 chest radiographs yielded an AUC of 0.95 (non-inferior to radiologist performance) in classifying microbiology-confirmed tuberculosis in external validation (see Figure 3.8 for more details).

Unsupervised transfer learning involves transferring knowledge between tasks that are similar, but with different data, and in which both the source and target tasks have unlabeled data. In some case, transfer learning without any labeled data from the target domain is also known as unsupervised transfer learning. Unsupervised transfer learning often involves techniques such as dimensionality

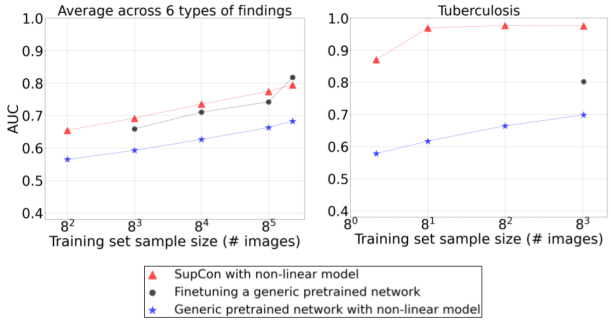

Figure 3.8: Efficacy of CXR (chest X-Ray) specific networks utilizing supervised contrastive (SupCon) learning compared to standard transfer learning from a non-medical dataset (red), with a control group using a generic pretrained network (blue). Performance is measured by the area under the receiver operating characteristic curve (AUC) for detecting various Chest X-Ray (CXR) abnormalities (left graph) and specifically tuberculosis (right graph). The SupCon approach, requiring significantly fewer labels, matches or surpasses the performance of the transfer learning model at equivalent dataset sizes, demonstrating up to a 688-fold reduction in label requirements. Notably, small nonlinear models trained on as few as 45 chest radiographs achieved an AUC of 0.95, comparable to radiologist assessment in identifying confirmed tuberculosis cases. Figures are taken from [61].

reduction and clustering to identify patterns in the data. In order to achieve that, we usually assume the existence of a common structure between source and target and leverage this information to perform the transfer [62].

Transductive transfer learning (also known as *domain adaptation*) refers to transferring knowledge between tasks with similar structure, but in which the source and target tasks have different data and the source task has a large amount of labeled data, while the target task has no labeled data (and usually following different distributions). In another term, the source and target domains have the same feature space while they can originate from different distributions. That is, the learning algorithm knows exactly on which examples it will be evaluated after training without any labelled samples in the target domain. This can become a great advantage to the algorithm, allowing it to shape its decision function to match and exploit the properties seen in the test set [63]. In this case, the model may use domain adaptation techniques to align the distribution discrepancy between domains in order to generalize the trained model to the domain of interest (adjust

to the different data distribution in the target task).

3.3.6 Parameter initialization

Initialization of neural network weights is an important factor that can impact the performance of a model during training. There are various methods for initializing weights, such as using constant values, sampling from a distribution, or using more sophisticated schemes like the so-called Xavier Initialization [64]. The authors of [64] demonstrated that networks initialized with Xavier achieved substantially quicker convergence and higher accuracy on the CIFAR-10 image classification task.

The choice of initialization method can affect the reproducibility and convergence speed of the neural network. The selection of initial values for the parameters of a neural network can significantly influence its performance. Inadequate initialization, such as random initialization of weights, may result in non-reproducibility and inferior performance, while initializing with constant values may delay convergence. Careful initialization can aid in learning with limited data, improve reproducibility, and optimize the training.

In a study comparing the effects of different weight initialization strategies on the performance of a CNN (Figure 3.9), it was found that initializing weights with values drawn from normal distributions with variances inversely proportional to the number of inputs into each neuron resulted in the best performance. The network achieved a validation accuracy of over 99% and a final loss two orders of magnitude smaller than networks initialized with weights set to zero (left plot on Figure 3.9) or drawn from normal distributions with a standard deviation of 0.4 (middle plot on Figure 3.9). These results suggest that careful weight initialization can significantly boost the convergence and performance of a neural network.

The effectiveness of transfer learning in dealing with small data problems can partly be attributed to parameter initialization since we basically try to pick the right initialization (initial checkpoints) pretrained on a similar but larger dataset. Hence we can state that the initial state of the parameters can significantly impact the optimization process, potentially leading to issues such as divergence, getting stuck at saddle points or local minima, and requiring a larger amount of training data

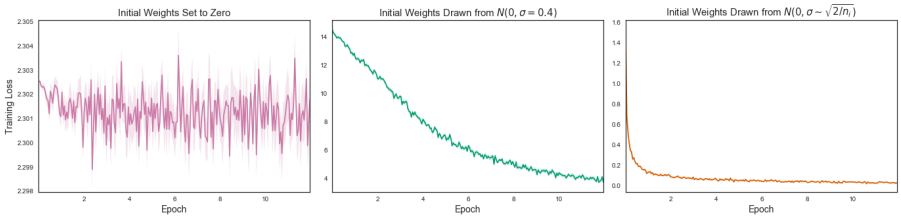

Figure 3.9: Comparative visualization of training loss trajectories using different weight initialization strategies for a CNN on the MNIST dataset. Each subplot displays the 10-batch rolling average of the loss incurred during the training of a basic CNN on the MNIST dataset, which consists of 60000 images of handwritten digits. The network was trained for a total of 12 epochs with a batch size of 128 images for each weight initialization strategy. The left plot shows the model with weights initialized to zero, exhibiting high variability and poor convergence. The middle plot illustrates weights initialized from a normal distribution with a standard deviation of 0.4, showing better, yet suboptimal performance. The right plot demonstrates the superior performance of weights initialized from a normal distribution with variance scaled inversely with the number of neuron inputs, yielding the most stable convergence and significantly lower loss. This strategy led to a network achieving over 99% validation accuracy and a notably smaller final loss compared to the other two methods, underscoring the importance of proper weight initialization in neural network training.(source: [65]).

for successful training. It is therefore important to carefully consider the initial parameter values chosen for a model.

3.3.7 Loss function reformulation

In modern DL research, the categorical cross-entropy loss after softmax activation is the method of choice for classification, that has not been questioned well enough. A recent paper, DL on Small Datasets without Pre-Training using Cosine Loss [66], claims to obtain around 30% increase in accuracy for small datasets when switching the loss function from categorical cross-entropy loss to a cosine loss (1 — cosine similarity) for classification problems.

The Cosine Loss function, which maximizes the cosine similarity between the output of a neural network and one-hot vectors [67] indicating the true class, has been found to be an effective method for learning from small datasets. This method has been shown to be superior to cross-entropy loss, possibly due to the inclusion of L2 normalization, which acts as a regularizer without the need for

additional hyperparameters. Experiments have demonstrated the effectiveness of this approach in small dataset scenarios (Figure 3.10).

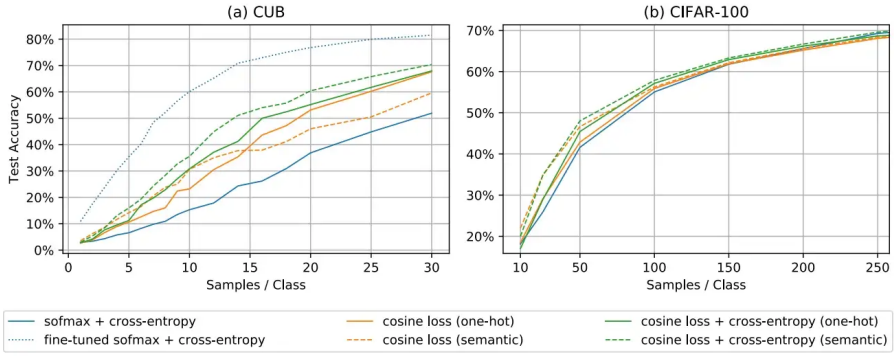

Figure 3.10: Test accuracy comparison between different loss functions on two datasets: (a) CUB (Caltech UCSD Birds) and (b) CIFAR-100. The graphs contrast the performance of softmax with cross-entropy loss and cosine loss with variants one-hot and semantic cross-entropy. The cosine loss, particularly with semantic cross-entropy, shows improved learning efficiency on the CUB dataset, a small dataset scenario, highlighting its advantage in scenarios with fewer samples per class. However, the performance gains are less distinct in larger datasets like CIFAR-100, indicating the loss function's relative impact based on dataset size. These findings suggest that cosine loss, due to L2 normalization acting as an implicit regularizer, is a robust choice for small datasets, outperforming the traditional cross-entropy loss in such contexts (source: [66]).

The Hinge loss function [68] has also been demonstrated to be effective in situations where resources are limited, allowing for the training of models on small datasets. In particular, the squared Hinge loss has been found to have faster convergence and improved performance compared to other variants of the Hinge loss. Additionally, it has been shown to be more resistant to noise in both the training set labels and the input space [69], [70].

3.3.8 Regularization techniques

Since the ML model trained with small data are prone to overfitting, regularization techniques can be considered an effective way of dealing with limited data in machine learning. Regularization is a technique used in ML to constrain the model fitting process and reduce the effective number of degrees of freedom without decreasing the actual number of parameters in the model. It is a popular method

to penalize overly complex models and prevent overfitting (penalizes the coefficients that cause the overfitting of the model), thus improving the generalizability of a model.

The penalty term is based on the magnitude of the model parameters, and its purpose is to discourage the model from fitting too closely to the training data. In the context of linear regression, regularization involves adding a penalty term to the cost function that is proportional to the magnitude of the coefficients. This forces the model to prefer solutions with smaller coefficients, which can help to mitigate overfitting and improve the model's ability to generalize to unseen data.

L1 Norm (Lasso) and **L2 Norm** (Ridge) regularization are two popular regularization techniques. In L1 regularization, the penalty term added to the cost function is the summation of absolute values of the coefficients, making the models with fewer non-zero parameters, which can be beneficial for the interpretability of model performance in a production setting. On the other hand, in L2 regularization, the penalty term added to the cost function is the summation of the squared value of coefficients, making the models with more conservative (closer to zero) parameters, which is similar to applying strong zero-centered priors to the parameters in a Bayesian framework. Generally, L2 regularization is more effective for improving prediction accuracy compared to L1 regularization.

Dropout [71] is a another method of regularization in which activations of randomly selected neurons are set to zero during the training process. This technique helps the network learn more robust features and reduces its reliance on the predictive power of a small group of neurons. [72] applied this concept to CNNs through the use of Spatial Dropout, in which entire feature maps are dropped out instead of individual neurons.

Weight decay (with AdamW - Adam with decoupled weight decay [73]), explained briefly in 3.3.12, is also one of the powerful techniques in regularization in which proper weight decay tuning help validation accuracy sometimes suddenly begins to increase from chance level toward perfect generalization, even long after severely overfitting on small datasets.

Batch normalization [74] can also be regarded as a regularization method that

normalizes the activations in a layer by subtracting the batch mean and dividing by the batch standard deviation. This technique is commonly used in the preprocessing of pixel values and has been demonstrated to improve the performance of CNNs. In addition to its regularization properties, batch normalization can also help stabilize the training process and reduce the sensitivity of the network to the choice of initialization parameters. [1] and [2] illustrated the impact of BN layer empirically under various settings and showed that the final BN layer, when placed before the softmax output layer, has a considerable impact in highly imbalanced image classification problems as well as undermines the role of the softmax outputs as an uncertainty measure. The impact of batch normalization on various settings will be explored in detail in Chapter 4.

3.3.9 Data augmentation

Data augmentation (DA) is a common technique used to virtually increase the size of a dataset by applying modifications to color, brightness, contrast, or adding noise to the existing data [75]. While this technique can be effective in improving the performance of ML models, it has the limitation of presenting the same samples in different forms to the model, which may not have a significant impact on generalization. Nevertheless, this can help to reduce overfitting. Due to its impact on by providing more stable and smoother response to various variations of input data, DA can even be considered as part of a broad set of regularization techniques aimed at improving model performance [76].

It is widely accepted in the field of ML that increasing the amount of data available, even if the quality is not optimal, can lead to improved model performance. Data augmentation can be effective because it incorporates prior knowledge into the dataset. The use of Generative Adversarial Networks (GANs) [77] and paired samples [78] are also being explored as methods for data augmentation.

DA techniques can be divided into two categories: data warping (basic image manipulations) and oversampling (Figure 3.11). Data warping techniques preserve the label of the original data and include transformations such as geometric and color manipulations, random erasing, adversarial training, and neural style transfer. Oversampling techniques, on the other hand, create synthetic instances and add

them to the training set. Examples of oversampling techniques include mixing images, feature space augmentations, and the use of GANs. It is worth noting that these categories are not mutually exclusive, and it is possible to combine techniques such as GAN sampling with data warping methods like random cropping to further increase the size of the training dataset [75].

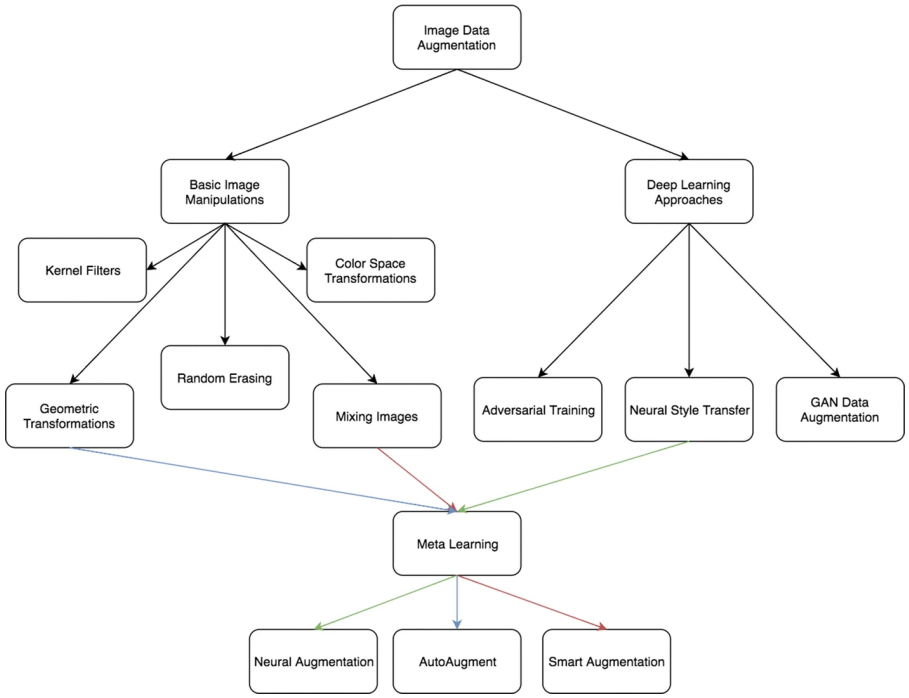

Figure 3.11: Image augmentation techniques (source: [75]).

GANs take random noise from a latent space and produce unique images that mimic the feature distribution of the original dataset. The model, based on image conditional GANs, takes data from a source domain and learns to take any data item and generalise it to generate other within-class data items. As this generative process does not depend on the classes themselves, it can be applied to novel unseen classes of data. There are a plethora of different types of GANs in the literature that can be used to generate synthetic data [79]. The comparison of basic image transformations and GAN-based augmentation can be seen at Figure 3.12.

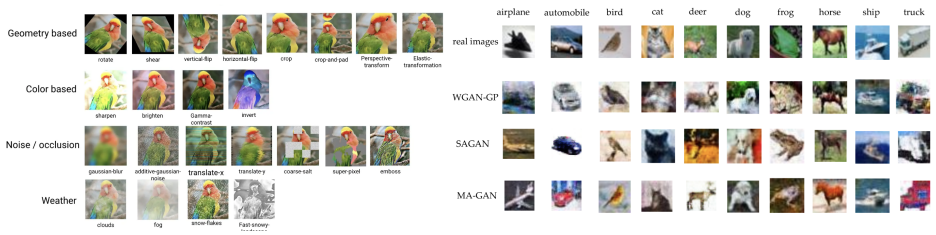

Figure 3.12: Basic image transformations [80] and GAN-based augmentation methods [81].

Paired sample is a surprisingly effective data augmentation technique for image classification tasks. In this technique, called SamplePairing [78], a new sample from one image is synthesized by overlaying another image randomly chosen from the training data (i.e., taking an average of two images for each pixel). This simple data augmentation technique significantly improved classification accuracy in various benchmark datasets .

DA can bring several benefits to ML models. One of the main advantages is the improvement of model prediction accuracy. By increasing the amount of training data available, models can be more robust and less prone to overfitting. Moreover, data augmentation can help mitigate data scarcity, which can be particularly useful in scenarios where data collection and labeling are costly. Additionally, data augmentation can also help address class imbalance issues in classification tasks by generating synthetic samples that help balance the distribution of classes in the dataset. Overall, data augmentation can increase the generalization ability of models, leading to better performance on unseen data.

As a result, DA can be used to address certain types of biases present in a small dataset. However, it is not a comprehensive solution and cannot create new categories of data that are not already represented in the dataset. Nevertheless, it can be effective in mitigating biases related to factors such as lighting, occlusion, scale, and background. Additionally, DA can help to prevent overfitting by artificially increasing the size and diversity of the dataset, which can have characteristics similar to those of a larger dataset. This can be particularly useful when working with limited data, as it can allow for the development of more robust models.

With the majority of DA techniques being manually created, recently there are various method developed to automate DA processes such as AutoAugment [82] and Trivial Augment [83] that achieve significant improvements on benchmark datasets.

3.3.10 Synthetic data generation

Synthetic data refers to artificially generated data that is designed to mimic real-world data. It can be used in a variety of contexts to preserve privacy in sensitive domains, such as medical and transactional data. Synthetic data can also be generated using a small amount of well-labeled data, and there are various techniques, such as SMOTE (Synthetic Minority Over-sampling Technique) [84], ADASYN (Adaptive Synthetic Sampling Approach) [85], Variational AutoEncoders (VAEs) [86], and the aforementioned GANs, that can be used to generate synthetic data. The use of synthetic data can facilitate the processing of large amounts of real-world data and accelerate the time and energy required for such processes.

SMOTE: One approach to addressing the issue of imbalanced data is generating synthetic data, which can be accomplished through techniques such as Synthetic Minority Over-sampling Technique (SMOTE) and Modified-SMOTE [87]. These methods generate new data points by taking the minority class data points and creating new points that lie between two nearest data points joined by a straight line in the feature space. The number of nearest neighbors used for data generation can be adjusted as a hyper-parameter based on the requirements of the problem. However, it is important to note that generating synthetic data can increase the risk of overfitting due to the presence of duplicate data.

One of the advantages of using SMOTE is that it mitigates the problem of overfitting caused by random oversampling, as synthetic examples are generated rather than replications of instances. Additionally, SMOTE does not result in the loss of useful information. However, a disadvantage of using SMOTE is that it does not take into consideration neighboring examples from other classes, which can result in an increase in overlapping of classes and the introduction of additional noise. Additionally, SMOTE may not be effective for high dimensional data (i.e. a dataset with large number of attributes or features). According to [88], high-dimensional problems are problems where the number of targeted features p is much larger

than the number of observations N, often written $p \gg N$.

The M-SMOTE algorithm is a modified version of the SMOTE method [87], which takes into account the underlying distribution of the minority class when generating synthetic data. This approach involves classifying minority class samples into three categories: security or safe samples, border samples, and latent noise samples. Security samples refer to those data points that have the potential to improve the performance of a classifier, while noise samples are those that may decrease the performance of the classifier. Data points that are difficult to categorize into either category are referred to as border samples. These categories are determined by calculating the distances between minority class samples and the training data. M-SMOTE then randomly selects a data point from the k nearest neighbors for security samples, selects the nearest neighbor for border samples, and does not generate synthetic data for latent noise samples.

Generative Adversarial Networks (GANs): Generative adversarial networks (GANs) are a class of neural networks [89] that consist of two sub-networks: a generator and a discriminator. The generator is responsible for synthesizing output data, while the discriminator is responsible for distinguishing between the synthesized data and real data. The generator and discriminator are trained to compete against each other, with the generator attempting to generate outputs that are indistinguishable from real data, and the discriminator trying to accurately identify whether an output is real or synthesized. Through this process, GANs are able to produce outputs that are highly realistic in appearance.

The use of GANs in data augmentation has received considerable attention due to their ability to generate new training data that leads to improved classification model performance. For instance, [90] used a large CT image database and trained a GAN to transform contrast CT images into non-contrast images; and then used the trained model to augment their training using these synthetic non-contrast images, and managed to reduce manual segmentation effort and cost in CT (computer tomograpght) imaging. In another study, [91] utilized public datasets and created synthetic versions using various GAN models. The effectiveness of these synthetic datasets as training data was evaluated through two methods. First, by comparing the accuracy, precision, and recall of a decision tree classifier trained on the original

data and one trained on the synthetic data. Surprisingly, in some instances, the classifier trained on synthetic data performed better than the one trained on the original data, indicating that GAN-based data augmentation can be a useful strategy to prevent overfitting. [92] applied GAN to approximate the true data distribution and generate data for the minority class of various imbalanced datasets and compared the performance of GAN with multiple standard oversampling algorithms.

[92] also argued that traditional methods such as zooming, cropping, and rotating are effective for object classification but are not applicable in cases like time series data presented in images where there is no singular object to classify. To enhance the classification accuracy through the deep neural network's capability to handle intricate data, the authors proposed a GAN-CNN classifier combination that showed a superior accuracy compared to the other ML methods [93].

The original GAN architecture, which utilizes multilayer perceptron networks in the generator and discriminator networks, is able to generate acceptable images for simple datasets like MNIST handwritten digits, but is not effective for producing high quality results for more complex, high resolution datasets. There have been numerous studies that have modified the GAN framework, including through the use of alternative network architectures, loss functions, and evolutionary methods, among others. These modifications have led to improvements in the quality of samples generated by GANs. Several new GAN architectures, including DCGANs [94], StackGAN [95], Progressively-Growing GANs [96], CycleGANs [97], and Conditional GANs [98], have been proposed and have been found to have potential applications in data augmentation.

3.3.11 Problem reduction

Problem reduction refers to the process of transforming a new or unknown problem into a known problem that can be easily solved using existing techniques. This approach has been demonstrated to be effective in scenarios where limited datasets are available. For instance, sound classification problem can be transformed into image classification by converting the voice clips into images (spectograms), which can then be addressed using state-of-the-art computer vision architectures and

techniques such as transfer learning. Research has shown that this approach can produce satisfactory results even with small datasets [99].

Another example would be transforming the image classification problem into two-stage problem such as object detection to extract patches and then image classification problem to classify the patches into target classes. This approach can be highly effective when there is not enough labelled images for the classification problems. Imagine we are trying to detect if a solar panel array (a group of solar panels that are connected together, collectively converting solar radiation into electricity) is defect or not; and the dataset is composed of a list of solar panel arrays each of which has only one class. Using problem reduction approach, rather than classifying the entire array into a single class, we can do the followings:

1. Crop the panels from the array using image processing and segmentation algorithms (Figure 3.13).

2. Using the original defect coordinates, assign positive defect label to the panel that contains these coordinates; and assign negative class (no defect) to all the other panels (now we have i, the number of defected panels, as the positively labelled panels, versus $(n - i)$, the complementary number of panels in an array, as the negatively labelled panels) (Figure 3.14).

3. Train a classifier on this dataset to find out defect panels.

4. In inference time, run segmentation algorithm at first to isolate the panels and then crop them.

5. Send each segmented panel into the classifier to detect if a panel has defect or not.

Another advantage of using this approach is reducing the problem complexity by filtering out the irrelevant portions of an image and only showing the useful parts. The learning capacity of a model trained on cropped patches is usually much higher compared to the one trained with entire images having many irrelevant parts and background noises.

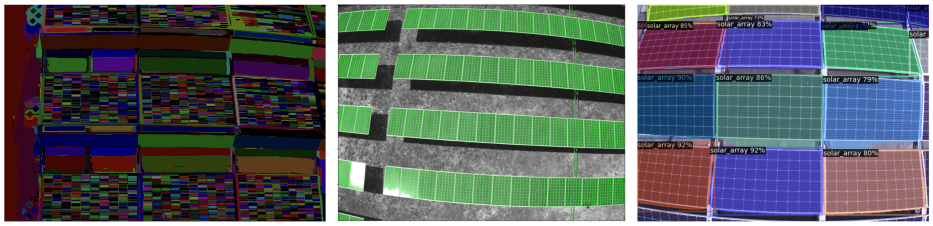

Figure 3.13: Segmenting and cropping the panels from solar array. Left: Global Contrast based Salient Region Detection (SGD) [100], middle: Enhanced line segment drawing (ELSED) [101], right: Detecting twenty-thousand classes using image-level supervision (DETIC) [102]. Images are taken from NORCE-PV dataset mentioned in [3].

3.3.12 Optimization techniques

The optimization of neural networks using gradient descent algorithms is significantly impacted by the size of the training dataset. While using smaller training sets, such as in the case of stochastic gradient descent, can result in faster training processes, updates may exhibit larger fluctuations. Conversely, training with larger datasets can be computationally expensive and slow. Therefore, the selection of the appropriate training dataset size is an important consideration in the optimization of neural networks.

Gradient descent (sometimes called batch gradient descent) is an example of an algorithm that performs better (in terms of computation speed) when the dataset is small, because it computes the gradients on the whole dataset in each iteration, as opposed to stochastic gradient descent which uses only a small part of the data in each iteration.

In the context of few-labelled samples, gradient-based optimization algorithms have been shown to be less effective due to two main reasons. Firstly, these optimization algorithms, such as momentum, AdaGrad, AdaDelta, and ADAM, are not specifically designed to perform well under the constraint of a limited number of updates. When applied to non-convex optimization problems, these algorithms do not provide strong guarantees of convergence speed, and may require a large number of iterations to reach a good solution. Secondly, the network must be

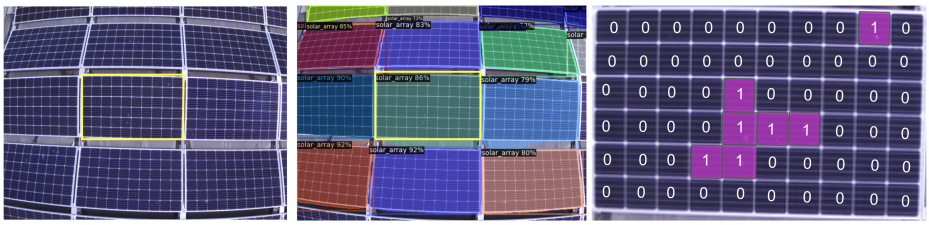

Figure 3.14: The solar panel array on the left image has 9 visible panels and only one of them is labelled as a defect. Rather than using this entire image in a 'solar array classification task,, segmenting & cropping the relevant panel at first (middle image), then each cell within a panel and then assigning a defect status labels ($0 = no\_defect, 1 = defect$) for each cell (image on the right), and then sending these $9x60 = 540$ patches to 'solar cell classification' task would result in a better performance as we will be ending up more data points to help the model learn better. The image on the right indicates the solar panel marked with yellow bounding box on the other images (left and middle). The original image is taken from NORCE-PV dataset mentioned in [3].

initialized with random parameters for each separate dataset, which hinders its ability to converge to a good solution after a few updates. Overall, these factors make it difficult for gradient-based optimization to effectively learn from small datasets [103].

In another study [37] that investigates the impact of optimization methods that work well under small data regimes, the authors show that long after severely overfitting, validation accuracy could suddenly begin to increase from random level toward perfect generalization (see Figure 3.4).

Normally, when using a supervised learning approach, reducing the amount of training data leads to a decrease in the model's ability to generalize when the optimization process is able to fit the training data perfectly. However, [37] found a different outcome: the model's performance remains constant at 100% within a certain range of training dataset sizes, but the time needed for optimization increases rapidly as the dataset size is reduced. They show empirically that different optimization algorithms lead to different amounts of generalization and the amount of optimization required for generalization quickly increases as the dataset size decreases. They compare various optimization details to measure their impact on data efficiency and find that *weight decay with AdamW* (Adam with decoupled

weight decay [73]) is particularly effective at improving generalization on the tasks they studied (see Figure 3.15). [73] illustrates that *Adam* generalizes substantially better with decoupled weight decay than with L2 regularization, achieving 15% relative improvement in test error on several image benchmark datasets.

Figure 3.15: Different optimization algorithms lead to different amounts of generalization within an optimization budget of 10^5 steps. Weight decay (with AdamW) improves generalization the most, but some generalization happens even with full batch optimizers and models without weight or activation noise at high percentages of training data. Suboptimal choice hyperparameters severely limit generalization (source: [37]).

In another study [104] investigating the best hyperparameter settings to couple self-supervised learning paradigms with semi-supervised techniques in low data regimes, the authors empirically demonstrate that weight decay is usually among the most important hyperparameters to tune while training deep neural networks, no matter what the size of the dataset is as depicted in Figure 3.16.

The "hypersweep curves" in Figure 3.16 depict the performance of a supervised baseline model on a fraction of the ILSVRC-2012 dataset, illustrating the impact of varying hyperparameters on model accuracy. The curves represent the sorting of models by accuracy with a fixed hyperparameter. For both figures, the rightmost point on each curve indicates the peak performance for that hyperparameter value. The spread between curves at these points reflects the sensitivity of the model to the hyperparameter, while the overall shape and proximity of the curves suggest

the robustness and interdependence of the hyperparameter values. The chart on the left uses the full custom validation set, whereas the other chart employs a reduced validation set with one image per class, which is shown to be sufficient for hyperparameter evaluation. Notably, the experiments reveal that weight decay and the number of training epochs significantly influence model training on limited data. Contrary to common belief, reducing model capacity by decreasing depth or width does not enhance performance; instead, deeper and wider models demonstrate superior performance and robustness, even with limited training data. These findings challenge prevailing assumptions and support recent observations that wider models can facilitate optimization.

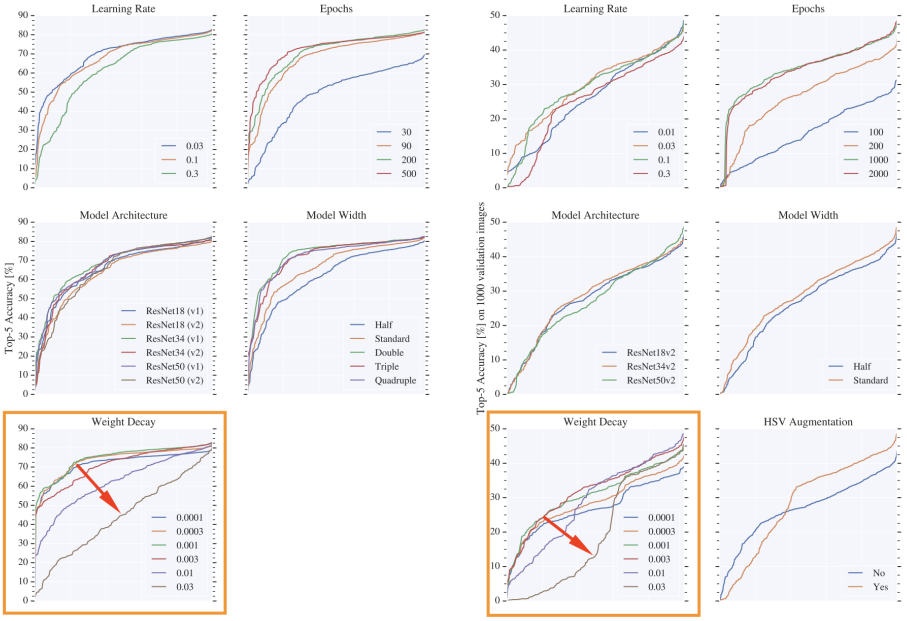

Figure 3.16: Performance impact of hyperparameter variation on a baseline model trained with a subset (10% on the left and 1% on the right) of the ILSVRC-2012 dataset. The curves demonstrate model accuracy against fixed hyperparameter values, with the largest spread between curves illustrating the pronounced effect of weight decay on model performance. These results underscore the significance of weight decay in training with limited data and challenge the notion that reduced model capacity benefits small dataset performance as well as indicating that weight decay is usually among the most important hyperparameters to tune while training deep neural networks as it may give the largest boost in validation accuracy compared to other hyperparameters (source: [104]).

3.3.13 Using physics-informed neural networks

The incorporating existing physical principles into ML can be a useful approach for modeling small data problems and developing more powerful models that learn from data and build upon our existing scientific knowledge. With the growth of ML and the availability of large amounts of scientific data, data-driven approaches have become increasingly prevalent in scientific research, including information science, mathematics, medical science, materials science, geoscience, life science, physics, and chemistry [105]. These approaches, also known as scientific ML (SciML) [106], do not require an existing theory and allow for the use of ML algorithms to analyze scientific problems solely based on data. This shift in the scientific method represents a departure from the traditional method of designing a well-defined theory and refining it through experimentation and analysis to make new predictions.

The physics-informed neural network (PINN) is a deep learning method that bridges the gap between machine learning and scientific computing by incorporating physical principles into ML. PINN has superior approximation and generalization capabilities, which made it gain popularity in solving high-dimensional partial differential equations (PDEs) [107], and has been used in various applications such as weather modeling, healthcare, and manufacturing.

PINN is able to predict the solution far away from the experimental data points, and thus performs much better than the naive network. PINNs can be easily applied to many other types of differential equations too, and are a general-purpose tool for incorporating physics into machine learning. The idea is adding the known differential equations directly into the loss function when training the neural network. This additional "physics loss" in the loss function tries to ensure that the solution learned by the network is consistent with the known physics [108].

Similarly, [109] also used a deep neural network to learn solutions of the wave equation, using the wave equation and a boundary condition as direct constraints in the loss function when training the network. By using the physics constraint in the loss function the network is able to solve for the wavefield far outside of its boundary training data, offering a way to reduce the generalisation issues of

existing DL approaches.

Figure 3.17 shows the effect of incorporating the harmonic oscillator formula into a scientific task to find a model that is able to accurately predict new experimental measurements given the first few measurements (observations). The problem is, using a purely data-driven approach (looking only at the actual values of the unknown physical process) cannot generalize beyond the observations. Once we incorporate the underlying physics formula into loss function, the model is able to generalize further beyond training dataset without using additional data points.

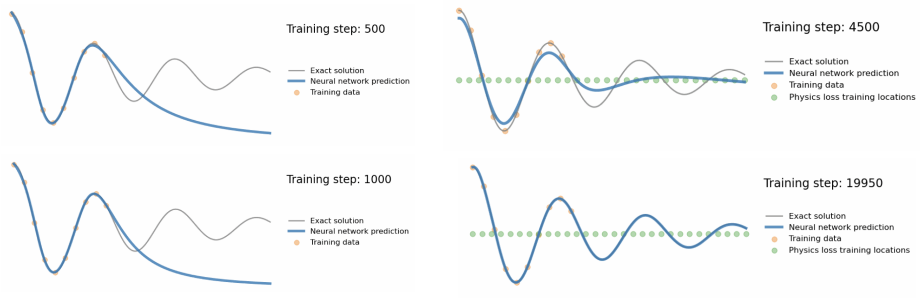

Figure 3.17: The physics-informed neural network is able to predict the solution far away from the experimental data points, and thus performs much better than the naive network. On the left hand side, even though the neural network can accurately model the physical process within the vicinity of the experimental data, it fails to generalise away from this training data. Once we incorporate the underlying physics formula into loss function, on the right hand side, the physics-informed network is able to generalize further beyond training dataset without using additional data points. Figures are taken from [110].

In short, PINN is a deep learning method that bridges the gap between machine learning and scientific computing by incorporating physical principles into ML. PINN has superior approximation and generalization capabilities, which made it gain popularity in solving high-dimensional partial differential equations (PDEs), and has been used in various applications such as weather modeling, healthcare, and manufacturing.

3.3.14 Unsupervised learning techniques

Unsupervised learning is a ML approach that aims to learn patterns in data without the use of labels. It relies on the model learning the structure of the input data based on the features present in the data. Feature extraction, which involves identifying and extracting relevant characteristics from the raw data, can often improve the performance of unsupervised learning approaches [111].

Common techniques used in unsupervised learning include clustering, which groups data points based on shared features, anomaly detection, which identifies data points that do not fit the distribution of existing examples, and manifold learning, a neural network approach that reduces the complexity of the data by learning latent variables such as Principal Component Analysis (PCA). Unsupervised learning can be applied to various types of data, including numerical and categorical data.

In the absence of labeled data, unsupervised learning techniques can be used to try to extract useful information from the dataset. For example, clustering algorithms can be used to group similar examples together, and dimensionality reduction techniques can be used to identify patterns and relationships within the data.

3.3.15 Semi-supervised learning

As the name suggests, Semi-supervised learning (SeSL) lies between the two extremes of supervised and unsupervised learning in terms of the availability of labeled data. In SeSL, the task is performed by utilizing both labeled and unlabeled datasets, with the aim of gaining a deeper understanding of the underlying data structure. Typically, SeSL is performed using a small labeled dataset and a relatively larger unlabeled dataset. The ultimate goal of this approach is to develop a predictor that can accurately predict future test data, surpassing the performance of predictors learned from labeled training data alone [112].

The application of SeSL techniques allows for the leveraging of labeled data, while also deriving structure from unlabeled data to improve the overall performance of the task. This is particularly beneficial in scenarios where the size of the labeled dataset is small. In such cases, traditional supervised learning algorithms are often prone to overfitting. However, by incorporating the understanding of structure

derived from unlabeled data during the training process, SSL effectively alleviates this issue. Additionally, SeSL methods provide an alternative solution to the challenge of building large labeled datasets for learning a task. These techniques are a step closer to mimicking the way humans learn, providing a more efficient and effective approach.

In recent literature, it has been shown that popular approaches to SeSL involve the introduction of a new loss term during training in a typical supervised learning setting. There are three main concepts that are typically used to achieve SeSL, namely Consistency Regularization, Entropy Minimization and Pseudo Labeling that allow SeSL to improve the performance of a supervised learning task when dealing with limited labeled data. Additionally, using generative models and graph-based methods in semi-supervised learning can be found in [113].

Consistency Regularization is a technique that aims to train a model that is robust to various data augmentations [114]. It enforces that the output of the predictor should not significantly change when realistic perturbations are made to the data points. This is achieved by minimizing the difference between the prediction on the original input and the prediction on the perturbed version of that input (decreasing the distances between features from differently augmented images). The idea behind this approach is to leverage the unlabeled data to find a smooth manifold on which the dataset lies. By achieving invariance to various data augmentations, the model can achieve robustness and generalize better to unseen data.

The most notable examples of Consistency Regularization include temporal ensembling (pi-model) [115], mean teachers (weight-averaged consistency targets) [116] and virtual adversarial training (VAT) [117] methods. Recently, Fan et al. [114] proposed a new perspective on the concept of consistency regularization, suggesting that instead of training a model that is invariant to all types of augmentations, it could be more beneficial to focus on improving equivariance on strongly augmented images, a simple yet effective technique, known as Feature Distance Loss (FeatDistLoss) that is designed to improve data-augmentation-based consistency regularization.

Entropy minimization [118] aims to encourage more confident predictions on unlabeled data. The model is trained to have low entropy predictions, regardless of the ground truth. Additionally, the confidence for all classes for an input example should sum to 1. This objective encourages the model to give high confidence predictions, and discourage the decision boundary from passing near data points where it would otherwise be forced to produce a low-confidence prediction.

The field of learning theory has traditionally focused on the two extremes of the statistical paradigm: parametric statistics, where examples are known to be generated from a known class of distribution, and distribution-free Probably Approximately Correct (PAC) [27] frameworks. However, Semi-supervised learning, which involves the use of both labeled and unlabeled data, does not fit neatly into these frameworks. This is because the usefulness of unlabeled data depends on the underlying distribution of the data, and no positive statement can be made without making distributional assumptions since unlabeled data coming from some distributions might be non-informative. This means that generalizing from labeled and unlabeled data may differ from transductive inference [118].

In this regard, the entropy minimization framework proposes an estimation principle applicable to any probabilistic classifier, aiming at making the most of unlabeled data when they are beneficial, while providing a control on their contribution to provide robustness to the learning scheme. This framework also shows that *unlabeled examples are mostly beneficial when classes have small overlap*, and provides a means to control the weight of unlabeled examples, and thus to depart from optimism when unlabeled data tend to hamper classification.

Pseudo-Labeling is a simple yet effective approach to achieve SeSL, and also known as proxy-labelling, self-training or co-training [119]. The method involves training a model on a labeled dataset, then using it (and based on some heuristics) to make predictions on unlabeled data. The examples from the unlabeled dataset where the model's prediction is confident (above a predefined threshold) are then selected and the predictions are considered as pseudo-labels. This pseudo-labeled dataset is then added to the original labeled dataset and the model is retrained on the expanded labeled dataset. This process can be repeated multiple times. Some examples of such methods are Self-training, Co-training and Multi-View Learning

[120].

Pseudo-Labeling is closely related to self-training, where the model is trained using its own predictions as labels. This approach allows for the efficient use of unlabeled data, as it can boost the performance of the model without the need for manual annotation. This technique is easy to implement and can be used in various applications where labeled data is scarce.

Recently, SeSL algorithms based on deep neural networks have been successful in achieving good results on standard benchmark tasks. However, it is argued that these benchmarks do not fully reflect the challenges that SeSL algorithms would face in real-world applications. In order to address this, [121] created a unified re-implementation of various widely-used SeSL techniques and tested in a suite of experiments. The results of these experiments indicate that the performance of simple baselines which do not use unlabeled data is often under-reported in the literature (see Figure 3.18).

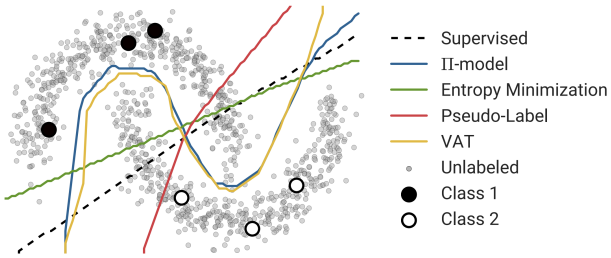

Figure 3.18: In the "two-moons" dataset, virtual adversarial training (VAT) and temporal ensembling (pi-model) methods were able to learn a highly accurate decision boundary with a relatively small amount of labeled data, specifically 6 data points depicted as large white and black circles in this figure. In contrast, Pseudo-Labeling was found to be inadequate in this context, resulting in the learning of a linear decision boundary. This highlights the limitations of Pseudo-Labeling as compared to VAT and pi-model in this specific dataset (source: [121]).

Additionally, the results show that different SeSL methods have varying sensitivity to the amount of labeled and unlabeled data. Moreover, the performance of these methods can degrade substantially when the unlabeled dataset contains examples that are out-of-distribution. These findings highlight the importance of rigorous

testing and evaluation of SeSL algorithms in real-world scenarios, in order to fully understand their capabilities and limitations [121].

In another study, [122] propose a new SeSL method called DP-SSL that adopts an innovative data programming (DP) scheme to generate probabilistic labels for unlabeled data. Different from existing DP methods that rely on human experts to provide initial labeling functions (LFs), they developed a multiple-choice learning (MCL) based approach to automatically generate LFs from scratch in SeSL style.

As a a result, it is essential to exercise caution when utilizing the technique of pseudo-labeling, as it has been observed that the model predictions can be fallacious at times. Furthermore, the model may generate several erroneous predictions for unlabeled data, leading to a detrimental feedback loop that may exacerbate the deterioration of performance. The same study also suggests that SeSL could be preferred by practitioners when there are no high-quality labeled datasets, labeled data is collected by sampling i.i.d. from the pool of unlabeled data and when the labeled dataset is large enough to accurately estimate validation accuracy.

For a detailed and comprehensive review of SeSL, the reader is referred to the Semi-Supervised Learning Book [112].

3.3.16 Self-supervised learning

Self-supervised learning (SSL) is a subcategory of unsupervised learning that utilizes unlabeled data to learn the representation of the data. This process is accomplished through a three-step process: generating input data and labels from the unlabeled data based on an understanding of the data, pre-training the model with the generated data and labels, and fine-tuning the pre-trained model for specific tasks of interest. In SSL, the objective is to learn generalizable and robust representations that can be applied to a variety of tasks and datasets, rather than simply learning high-level features.

Through self-supervised learning, it is possible to generate pre-trained backbone networks to extract features from domain-specific images and convert them into numerical vectors known as embeddings. This allows for the creation of models with fewer data and computational resources within that particular domain. In

some cases, even with less labelled data, this approach has enabled performance comparable to state-of-the-art DL models across various prediction tasks.

The most salient similarity between self-supervised and semi-supervised learning is that both approaches do not rely entirely on manually labeled data. However, this is where the similarity ends in broader terms. Self-supervised learning relies on the inherent structure of the data to make predictions, and does not require any labeled data (the system learns to predict part of its input from other parts of it). In contrast, semi-supervised learning still utilizes a limited amount of labeled data to guide the model's learning process.

Self-supervised learning is a primary component of this dissertation and will be elaborated in detail in Chapter 5.

3.3.17 Zero-shot, one-shot and few-shot learning

Zero-shot learning (ZSL) is a ML method in which a model is able to complete a task (classify and predict the class of an unseen sample, or detect an unseen object) without having received any training examples. It is a variant of transfer learning, where the model is trained on a set of classes and is able to generalize to unseen classes by leveraging additional knowledge about those classes, such as their textual descriptions or attributes. Unlike transfer learning, ZSL involves transferring knowledge between disparate feature and label spaces.

ZSL can be particularly useful when the number of classes in a dataset is large and obtaining labeled data for all classes is infeasible or impractical. It has garnered significant attention in recent years due to its ability to perform classification tasks with limited annotated data. This approach has been applied to various domains, including healthcare for medical imaging and COVID-19 diagnosis using chest x-rays, as well as unknown object detection in autonomous vehicles. As research continues to focus on methods that utilize minimal data with minimal annotation, the potential applications for zero-shot learning will likely expand.

In ZSL, when no labeled instances of the target classes are present in the training data, auxiliary information is utilized to facilitate the classification of unseen classes. This auxiliary information may take the form of class descriptions, known

attributes, semantic information, or word embeddings, and is necessary in order to effectively solve the ZSL problem. Through this process, ZSL techniques are intended to learn intermediate semantic layers and their properties, then apply them to predict a new class of data at inference time. For example, if we have a model that has been trained on images of horses and is presented with an image of a zebra, it might be able to recognize the zebra as a novel class based on its similarity to the known class of horses and the additional information that zebras have black and white stripes.

In ZSL, a labeled training set of seen classes and knowledge about the semantic relationships between seen and unseen classes is used to classify instances belonging to unseen classes. This is achieved through the use of a high-dimensional vector space, called the semantic space, in which the knowledge from seen classes can be transferred to unseen classes. The process of ZSL typically involves learning a joint embedding space in which both semantic vectors (prototypes) and visual feature vectors can be projected, and using nearest neighbor search to match the projection of an image feature vector with that of an unseen class prototype in this space [123].

Few-shot learning, also known as low-shot learning, on the other hand, is a method for classification when there are only a few examples per class. It is typically implemented in the form of N-way k-shot problems, where there are N classes and k labeled examples for each class. This approach utilizes meta-learning across a range of classification tasks in order to quickly adapt to a new task. The meta-learning model is trained on a large set of N-way k-shot tasks drawn from a similar dataset. Few-shot learning is simply an extension of ZSL, but with a few examples to further train the model.

Similarly, one-shot learning involves the use of only one instance or example of data for training, as humans have the ability to learn even with a single example and are still able to distinguish new objects with high precision. This approach is particularly useful in tasks such as identifying an image of a person in identification documents, where a large dataset may not be available. One approach to address this challenge is to modify the loss function to be more sensitive to subtle differences in the data and learn a better representation of it. Siamese networks are a commonly

used method for image verification in one-shot learning.

3.3.18 Meta learning

Meta-learning, also known as learning to learn, is a ML approach that aims to learn how a ML model learns and use this knowledge to improve the training process. By being trained on a variety of similar tasks, a meta-learner can learn the most effective way to learn an unseen task. Meta-learning has the potential to be particularly useful for limited data problems, as it can enable the rapid adaptation to a new environment and generalization to unseen tasks with only a few examples. This is achieved by learning a high-level representation of the learning process itself, allowing the model to adapt to new tasks more efficiently.

Meta learning algorithms can achieve superior generalization performance compared to non-meta learning algorithms, even in the presence of small data. However, meta-learning requires a large number of tasks from a similar dataset for meta-training. Once the meta-model is trained, it can be applied to learn an unseen task with a small amount of data, and can also support lifelong learning by continuously improving as it is being used.

In meta-learning, the aim is to learn the learning process itself, rather than simply recognizing patterns in training data and generalizing to unseen data, as is the goal in conventional supervised learning. During the meta-training phase, the algorithm learns to identify similarities and differences between examples from different classes in the training set. In each iteration, a support set containing n labeled examples from k classes is used along with a query set consisting of previously unseen examples from unknown classes. During training, the loss function assesses the performance on the query set, based on knowledge gained from its support set and will backpropagate through these errors.

One of the most successful meta-learning algorithms that is designed to work well with limited data is known as Model-Agnostic Meta-Learning (MAML). The key idea underlying this method is to train the model's initial parameters such that the model has maximal performance on a new task after the parameters have been updated through one or more gradient steps computed with a small amount of data from that new task [124].

Meta-learning can be conceptualized as an optimization problem, where a second network is used to predict the parameters for a given task, or where an initialization for a neural network is learned. Additionally, meta-learning can be approached through the use of similarity among features in examples, by embedding these examples into a vector space and using a metric for classification. Finally, meta-learning can also be achieved through the incorporation of prior knowledge about the structure of naturally occurring tasks, in order to improve model performance.

3.3.19 Harnessing model uncertainty

In ML and statistics, traditional methods for modeling uncertainty rely on probability theory. However, it has been argued that conventional approaches to probabilistic modeling, which capture knowledge in terms of a single probability distribution, fail to distinguish between two distinct sources of uncertainty, referred to as *aleatoric* and *epistemic* uncertainty. Aleatoric uncertainty, also known as statistical uncertainty, refers to the notion of randomness, or the variability in the outcome of an experiment that is due to inherently random effects. An example of this type of uncertainty is coin flipping, where the data-generating process has a stochastic component that cannot be reduced by any additional source of information. In contrast, epistemic uncertainty, also known as systematic uncertainty, refers to uncertainty caused by a lack of knowledge (about the best model). This type of uncertainty is caused by ignorance of the agent or decision maker, and can in principle be reduced on the basis of additional information [125]. Knowing the type of uncertainty during model development is a crucial component and helps the modeler avoid seeking more data if it is an aleatoric uncertainty in which the uncertainty is irreducible even if more data is provided.

In other words, epistemic uncertainty refers to the reducible part of the total uncertainty, whereas aleatoric uncertainty refers to the irreducible part. Recognizing and distinguishing between these two types of uncertainty allows for more nuanced and accurate modeling of uncertainty in a wide range of applications. In some cases, both aleatoric and epistemic uncertainties might be present, but it can be challenging to determine which type of uncertainty should be associated with a specific phenomenon during the modeling phase. This distinction is important as it affects the way we model the uncertainty and model, while it's not always

well defined. In some cases, it may be necessary to gather more data or perform additional experiments to accurately categorize the source of uncertainty [126].

When the sample size is small, specifying the right model class is not an easy task, so the model uncertainty will typically be high due to the lack of evidence in favor of any class (an inherent problem especially with semi-supervised learning wherein we need to define a threshold that decides to which class a data point belongs).

In an ideal scenario, even with powerful models that can generalize effectively beyond test sets, it is expected that a degree of uncertainty would exist for samples that are entirely unknown. This uncertainty can be leveraged to estimate areas in which the model may not have sufficient knowledge. This is particularly important in low data regimes, where it is plausible that not all parts of the distribution are equally represented. In such cases, it is important that the model's predictions for unseen samples return a degree of uncertainty that can be utilized for various purposes, such as reducing false positive rates, identifying predictions that require human intervention, or as a threshold in semi-supervised or active learning approaches.

Thus, it is important to understand and model the uncertainty present in the problem in order to know when the model's predictions can be trusted. One important aspect of this is considering the confidence of the predictions made by the model in addition to the predictions themselves. To achieve this, generating confidence intervals for evaluation metrics when comparing different models can be helpful in preventing erroneous conclusions from being drawn.

This is particularly important when working with small datasets, as certain regions of the feature space may be underrepresented. In such cases, it is better to predict with a margin of error, rather than point estimates. Although models on small data will have large confidence intervals, being aware of the range of predictions can be beneficial when making actionable decisions. By taking into account the uncertainty present in the problem, we can make more informed decisions about the trustworthiness of the model's predictions.

There are two other important and relevant (tangential) issues with deep learning. First, the model might be overconfident even if it is gravely wrong. Second, the model may still try to assign a confidence score to out-of-distribution samples. For

example, if a user feeds an image of a healthy chest X-ray to a Covid classifier model, the model should return a prediction with a high level of confidence. However, when a user uploads an image of head CT and asks the model to predict the Covid status, the model is faced with a situation of out-of-distribution test data. The model has been trained on images of X-rays, but has never seen a head CT before. The image of the head CT lies outside of the distribution of data the model was trained on.

This example can be extended other settings, such as MRI scans with structures that a diagnostic system has never observed before, or scenes that an autonomous car steering system has never been trained on. In such cases, a desirable behavior for the model would be to return a prediction, but also to convey that the point lies outside of the data distribution and therefore, the model possesses some quantity conveying a high level of uncertainty with such inputs [52].

Understanding the model uncertainty also plays a crucial role in an active learning framework. By recognizing which unlabelled data points would be the most beneficial to learn from, the model can make informed decisions on which data points to request labels for from an external resource such as a human annotator. These data points are selected using an acquisition function, which evaluates the potential value of each point for learning. Various acquisition functions exist, many of which factor in the model's uncertainty about the unlabelled data to make these decisions. By utilizing this approach, it allows for a more efficient use of limited labelled data, resulting in a more robust model [127].

3.3.20 Active learning

Active learning is a semi-supervised approach that can be leveraged when large amounts of data are present but obtaining labeled data is costly. Instead of randomly labeling data, active learning allows teams to strategically select the data points that will have the greatest impact on the performance of the model. In other words, the key idea behind active learning is that a ML algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns (i.e. with a limited number of training labels by actively selecting data for learning). The process is outlined in the active learning loop

in Figure3.19. In each iteration, the ML model is trained with a growing dataset created by labeling new data selected from a pool of unlabeled data.

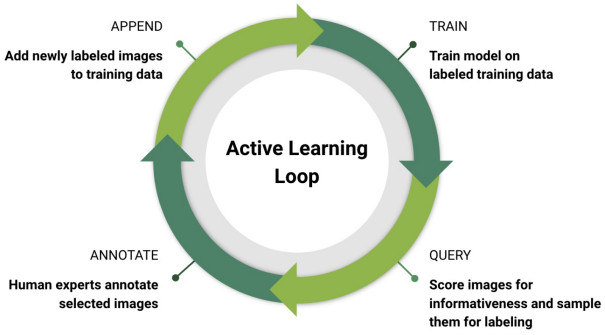

Figure 3.19: The Active learning loop diagram depicts the process of incrementally increasing the training dataset through selective labeling. In each iteration, the query step utilizes a scoring function and sampling strategy to determine which images should be added to the training dataset for further training after being labeled (source: [128]).

One way to implement active learning is by utilizing ML models for conducting preliminary tasks. These models can identify samples that are hard to classify, and then a human annotator can focus on labeling only those samples (selecting the next best data points to label). This way, the model can learn from the most informative data points, leading to more accurate predictions.

To further illustrate the concept of active learning, consider the analogy of a teacher and a student. Passive learning is like a student sitting in a class, listening to the teacher's lecture without engaging. On the other hand, active learning is when the student is an active participant, asking questions and collaborating with the teacher. The teacher in this scenario, focuses on the student's needs and spends more time on the concepts that are hard for the student to grasp. In this way, both the student and teacher are actively engaged in the learning process.

Similarly, in ML development, active learning is a collaborative process between the annotator and the modeler. The annotator provides a small labeled dataset, and the modeler uses it to train the model. The model then generates feedback on which data points should be labeled next. These models can identify samples that

are hard to classify, and then a human annotator can focus on labeling only those samples. This way, the model can learn from the most informative data points, leading to more accurate predictions. Through several iterations, the team can develop a more accurate model and a labeled gold training set.

Active learning is a training strategy that aims to reduce the amount of data that requires human labeling by intelligently (actively) selecting the most informative examples for labeling [129] where 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. This is accomplished through the use of a query strategy, which is applied to a ML model that has been initially trained on a small number of seed labeled examples. The process of labeling additional examples and updating the model is then repeated until the model is adequately trained. Active learning has been applied to a variety of ML tasks, including classification, detection, segmentation, and regression.

This approach is different from classical passive learning, where examples to be labeled are chosen randomly from the unlabeled pool. Instead, active learning aims to carefully choose the examples to be labeled in order to achieve a higher accuracy while using as few requests as possible, thereby minimizing the cost of obtaining labeled data. This approach is of particular interest in problems where data may be abundant, but labels are scarce or expensive to obtain. By carefully selecting the examples to be labeled, active learning allows to make the most of the available data while minimizing the cost of obtaining labeled data (keeping the annotation efforts at a minimum), making it useful in various real-world applications where labeled data is scarce and expensive [113].

In order to effectively identify difficult data points in active learning, a various combinations of methods are used to select the most informative samples for annotation (i.e. the process of identifying the most valuable examples to label next). These methods include classification uncertainty sampling, margin uncertainty, entropy sampling, disagreement-based sampling, information density, and business value [129]. An example of such a method can be found in the object detection problem. In this context, [128] proposed an image level scoring function that evaluates the usefulness of each unlabeled image for training. The selected images,

after being labeled, are then added to the training set. The experiments demonstrate the effectiveness of this approach, with up to 4 times relative mean average precision improvement compared to manual selection by experts.

Classification uncertainty sampling involves selecting the samples with the highest uncertainty, or the data points that the model knows the least about. By labeling these samples, the model becomes more knowledgeable and its performance improves [130].

Margin uncertainty, on the other hand, involves selecting the samples with the smallest margin. These are data points that the model knows about but is not confident enough to make good classifications. Labeling these examples increases the model's accuracy [131].

Diversity sampling aims to gather a representative sample of the entire data distribution, recognizing the significance of diversity as the model should perform well on any data encountered, not just a limited subset. The selected samples should reflect the underlying distribution. Common methods frequently depend on measuring the relationship between samples [132].

Entropy sampling uses entropy as a measure of uncertainty, which is proportional to the average number of guesses one has to make to find the true class. In this approach, we pick the samples with the highest entropy [130].

Disagreement-based sampling focuses on those samples where different algorithms disagree (asking annotators to label the examples on which classifiers disagree) [133]. For measuring the level of disagreement, there are several approaches used such as Kullback-Leibler (KL) divergence [134] and Jensen-Shannon divergence [135].

The **information density method** focuses on a denser region of data and select few points in each dense region. Labeling these data points help the model classify large number of data points around these points [136].

Lastly, the **business value method** focuses on labeling the data points that have higher business value than the others. This approach helps to prioritize labeling

the samples that are most important for the specific use-case and improve the model performance in real-world scenarios [137].

In active learning, the most informative examples are the ones that the classifier is the least certain about (a model has the least certainty are often the most challenging examples to classify). These examples are typically located near the class boundaries, and as a result, provide valuable information about the boundary between different classes.

In the field of ML, it is commonly understood that examples for which a model has the least certainty are often the most challenging examples to classify. These examples are typically located near the class boundaries, and as a result, provide valuable information about the boundary between different classes.

For instance, consider a binary classification problem in the medical domain where the task is to differentiate between benign and malignant tumors. The model may have a high level of certainty when presented with a clear and well-defined benign tumor, but may struggle to classify a malignant tumor that has similar features to a benign one. In this scenario, the model is uncertain about the correct classification of the example, as it lies near the class boundary between benign and malignant tumors. By observing this difficult example, the model can gain valuable information about the features that distinguish benign and malignant tumors and improve its performance on similar examples in the future.

Furthermore, research [129] has shown that models that are trained on a diverse set of examples, including difficult examples near class boundaries, tend to perform better on unseen data compared to models that are trained on a more homogeneous set of examples. Thus, it is crucial for the learning algorithm to identify and observe the difficult examples in order to gain a deeper understanding of the underlying data distribution and improve its overall performance.

In summary, the intuition behind the importance of difficult examples in ML is that they provide valuable information about the class boundaries and improve the performance of the model on unseen data. It is essential for the learning algorithm to identify and observe these difficult examples in order to gain a deeper understanding of the underlying data features.

Detailed overview of active learning concepts and methods can be found at [129].

3.3.21 Self-learning

Self-learning is a method that lies between semi-supervised and weakly supervised learning [138], utilizing an existing classifier to generate pseudo-labels for unlabeled data to train a new model. It operates by iteratively learning a classifier by assigning pseudo-labels to a subset of unlabeled training data with a margin greater than a set threshold. These pseudo-labeled samples are then combined with labeled training data to train a new classifier. The goal of pseudo-labeling is to generate labels for unlabeled data and improve the model's training with more information than before [139].

In contrast to self-supervised learning, where the model relies on the underlying structure of data to predict outcome (does not utilize labeled data), self-learning use both labeled and unlabeled data allowing the model (or models) to learn from themselves (or each other).

Another type of self-learning is called two-classifier self-training, which is similar to pseudo-labeling but utilizes two classifiers instead of one. The process involves training a classifier on the available data, then making predictions on the next batch of new data, alternating the classifier being used several times. This approach aims to improve the model's robustness by continuously feeding the output of one classifier as input to the other (i.e. each model learns on the output of the other) [140].

3.3.22 Multi-task learning

Multi-Task Learning (MTL) is a ML technique that leverages information from related tasks to improve performance on a primary task by training a model with multiple losses to perform well on multiple tasks (i.e., more than one loss function is optimized) [141].

In traditional ML, a single model or ensemble of models is trained to optimize for a specific metric, such as a benchmark score or a business KPI. However, by focusing solely on this single task, valuable information from related tasks is ignored. MTL

aims to overcome this limitation by sharing representations between related tasks, resulting in better generalization on the primary task.

For instance, in healthcare, a model trained to predict multiple medical conditions based on patient data can utilize information from related tasks, such as identifying risk factors, to improve predictions for each condition. Likewise, in agriculture, a model trained to predict crop yields for multiple crops can share representations between tasks, such as identifying soil characteristics, to improve predictions for each crop.

From a biological perspective, MTL can be considered modeled after human learning. In this context, when we learn new tasks, we tend to utilize the skills we have gained from related tasks. For instance, a baby starts by developing the ability to recognize faces and then applies that skill to identifying other objects.

MTL works effectively by incorporating two important concepts: Implicit data augmentation and Attention focusing [141].

Implicit data augmentation refers to the idea that by training a model on multiple tasks simultaneously, the model is effectively exposed to a larger sample size, leading to a better generalization of the model. This is because different tasks often have different patterns of noise, and training a model on multiple tasks helps average out these patterns and learn a more general representation.

Attention focusing is another advantage of multi-task learning. In limited and high-dimensional datasets or noisy tasks, it can be difficult for a model to identify relevant features. By training on multiple tasks, the model is able to focus its attention on important features as the additional tasks provide evidence for the relevance or irrelevance of these features.

When labeled data for the desired task is not present, an alternative approach in MTL is to utilize a task that is opposite of the goal. This can be accomplished through an adversarial loss that maximizes training error by using a gradient reversal layer. The adversarial task is predicting the input's domain, and by maximizing the adversarial task loss through gradient reversal, the model is forced to learn representations that are indifferent to the domains, which altogether

benefits the main task [141].

In summary, MTL provides a way for a model to learn more general representations and focus on important features that might not be easy to learn just using the original task, hence leading to improved performance on multiple tasks.

3.3.23 Symbolic learning

Symbolic learning is a form of ML that leverages symbolic representations of knowledge to make predictions with limited training data. By incorporating human-curated information into the learning process, this approach can effectively minimize the amount of training data required while enhancing the reliability and robustness of the ML system. This integration of human knowledge and ML capability also enables the creation of explainable ML systems, providing an opportunity to leverage a wealth of human knowledge to achieve performance outcomes that were not previously possible. Furthermore, this type of ML enhances the interaction between humans and ML systems by making ML's decisions more understandable to humans [142].

Symbolic models are fusing a representation of contextual knowledge into ML algorithms to improve algorithm performance [143]. These models utilize a representation of knowledge, often human-curated, such as an ontology, database, or contextual information. Traditional ML algorithms can be viewed as learning a representation of the features in the classes; in contrast, the symbolic model approach aims to reduce the amount of data required by introducing a human-constructed representation (knowledge).

Here are some examples:

- **Diagnosing diseases:** Imagine a patient presents with symptoms such as a cough, fever, and fatigue. A doctor might use his/her knowledge of common diseases to diagnose the patient with a respiratory infection. Similarly, symbolic learning algorithms can use information from medical ontologies or databases to diagnose diseases based on symptoms and other contextual information.

- **Predicting drug interactions:** When a patient is prescribed multiple medications, it is important to ensure that they do not interact in harmful ways. A pharmacist might use their knowledge of common drug interactions to determine the safety of a particular combination of drugs. Symbolic learning algorithms can be trained on data from drug databases to predict potential drug interactions and suggest alternatives if necessary.

- **Identifying risk factors for chronic diseases** Chronic diseases such as diabetes and heart disease are often the result of multiple risk factors, such as obesity, lack of exercise, and smoking. A doctor might use their knowledge of these risk factors to predict a patient's likelihood of developing a chronic disease. Symbolic learning algorithms can be trained on data from medical studies to identify and rank the most important risk factors for various chronic diseases.

- **Detecting plant diseases:** Diseases can significantly reduce crop yields and have a major impact on agriculture. Symbolic learning algorithms can be trained on data from plant disease databases to detect the presence of specific diseases based on symptoms such as discoloration of leaves or wilting of stems.

In each of these examples, symbolic learning is used to combine the strengths of ML algorithms and human knowledge to improve the accuracy and efficiency of decisions.

The selection of training samples is essential to any symbolic modelling efforts and not all data samples are equally informative: some carry unique information about the system, while others are redundant. To that end, [144] proposed an approach for constructing compact training data sets that serve as an input to a model learning method. For model learning, symbolic regression is chosen due to its ability to construct accurate models in the form of analytic equations even from small data sets as it, symbolic regression, outperforms other models for small data sets [145].

In summary, symbolic learning leverages human-curated information to perform limited data learning tasks and reduce the amount of data required for training. It

is a way of combining the strengths of traditional machine learning algorithms and human knowledge to improve the accuracy of predictions.

3.3.24 Hierarchical learning

Hierarchical learning is a ML approach that takes advantage of the hierarchical structure of real-world categories and taxonomies (i.e. using a taxonomy representing the relationship between objects and higher level classes) [146]. The hierarchical learning concept involves building a series of classification models that are structured based on a predefined hierarchy. The method utilizes transfer learning to build subsequent models, where the output of one model serves as input for the following. For instance, in healthcare, we can use hierarchical learning to diagnose diseases based on symptoms. A patient with a headache and nausea could be diagnosed with a migraine, but the model can also express uncertainty about the specific type of migraine (e.g., menstrual migraine or vestibular migraine). The hierarchical structure helps to make more accurate diagnoses based on limited data. In agriculture, hierarchical learning can be used to identify and classify different types of crops. For example, a model can classify a plant as a type of fruit, and then further classify it as an apple or a pear based on the shape, color, and other attributes of the fruit. This hierarchical structure helps to manage uncertainty in the classification process and makes it possible to train classifiers even when there is limited data available for certain types of crops.

[147] proposed a two-step framework using hierarchy transfer learning to build deep learning models for disease detection and classification, which achieved high accuracy and improved performance compared to other training approaches. The framework was extended to handle multiple input images and improved accuracy was achieved using a stacking ensembled method, leading to an improved performance even with a limited number of images.

3.3.25 Knowledge distillation based learning

Knowledge Distillation is a technique utilized to transfer the knowledge acquired by a complex model to a simpler model, with the goal of achieving similar or better performance on unseen data [148]. One of the key advantages of Knowledge Distillation is its ability to compress the knowledge of an ensemble of large base

models into a single, simpler model. This is particularly useful in situations where the storage and computational resources required by the ensemble of models is prohibitive. By distilling the knowledge of the complex model into a smaller, more tailored model, the efficiency and effectiveness of the model can be improved.

For example, when using a model trained on a dataset such as ImageNet, which contains 14 billion images and 100 classes, as a binary classifier for cats and dogs in a specific application, the computational resources required would be excessive. However, by using this model as a teacher and distilling its knowledge into a simpler model specifically designed for this task, the efficiency and effectiveness can be greatly improved.

To effectively transfer knowledge, it is important to examine the data used to train the network as it focuses on a specific area instead of the entire input space. However, access to this data may be restricted due to privacy concerns in various industries such as medicine, military, and industrial. In order to address this issue, [149] suggested KEGNET (Knowledge Extraction with Generative Networks), a new approach for knowledge distillation that does not require access to the original data. KEGNET learns the relationship between data points through training generator and decoder networks and generates artificial data to estimate the missing data on the manifold. At the same time, [150] argues that Knowledge Distillation is not as effective as widely believed, as there is often a significant gap between the teacher and student's predictive distributions, even when the student has the ability to exactly imitate the teacher.

3.4 Dealing with imbalanced data

In the scientific literature, data imbalance refers to the situation in which the number of observations in one class significantly exceeds those in other classes. This issue is often encountered in tasks such as detecting anomalies in electricity usage or identifying rare diseases, where the minority class (anomalies or rare cases) is of particular interest. If conventional ML algorithms are applied to imbalanced data without addressing this issue, the resulting models may be biased and inaccurate, as these algorithms are typically designed to minimize error assuming the class distribution is well-representing and unbiased. Examples of business problems

with imbalanced datasets include detecting rare diseases in medical diagnostics, identifying customer churn in the telecommunications industry, and predicting natural disasters such as earthquakes.

The problem of unbalanced data, in which the frequency of minority class observations is significantly lower than that of the majority class, can be addressed by balancing the data through sampling techniques. This can be achieved through increasing the frequency of the minority class (oversampling) or decreasing the frequency of the majority class (undersampling). The choice of oversampling versus undersampling and random versus clustered sampling depends on the size of the overall dataset and the distribution of the data. In general, oversampling is preferred when the dataset is small, while undersampling is more suitable when the dataset is large. Similarly, the decision between random and clustered sampling is influenced by the distribution of the data.

Oversampling (up-sampling): One approach to addressing imbalanced data is over-sampling, which involves increasing the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample. This method has the advantage of not leading to information loss, and it has been shown to outperform under-sampling. However, it also has the disadvantage of increasing the likelihood of overfitting due to the replication of minority class events.

Another up-sampling technique is K-means clustering that is independently applied to minority and majority class instances. This approach involves identifying clusters in the dataset and oversampling each cluster such that all clusters of the same class have an equal number of instances and all classes have the same size. While this technique has the advantage of addressing both inter- and intra-class imbalances, it also carries the risk of overfitting the training data. Overall, K-means clustering is a useful method for addressing imbalanced data, but it should be used with caution to avoid compromising the generalizability of the model.

Undersampling (down-sampling): In the context of addressing imbalanced data, undersampling involves randomly eliminating majority class examples in order to balance the class distribution. While this technique can be useful for improving run time and storage issues when dealing with large training datasets,

it is also important to consider its potential drawbacks, such as the potential for discarding potentially useful information and the risk of introducing bias through the random selection of samples. Additionally, the resulting sample may not accurately represent the overall population, leading to potentially inaccurate results when applied to the actual test dataset.

There are other techniques to allow the model learn under extreme imbalanced datasets. A concrete technique that adds a final batch normalization layer is one of this work's contributions and will be explored in detail in Chapter 4.

3.5 Anomaly Detection as a Small Data Problem

Anomaly detection, also known as outlier detection or novelty detection, is the task of identifying unusual or abnormal data points in a dataset. It is a crucial task in a wide range of applications, including fraud detection, cybersecurity, and quality control.

Detecting anomalies that are hardly distinguishable from the majority of observations is a challenging task that often requires strong learning capabilities. Since anomalies appear scarcely, and in instances of diverse nature, a labeled dataset representative of all forms is typically unattainable. In some cases, such as working with underrepresented populations or studying rare medical conditions, only limited data are available [61].

Despite tremendous advances in computer vision and object recognition algorithms, their effectiveness remains strongly dependent upon the size and distribution of the training set. Real-world settings dictate hard limitations on training sets of rarely recorded events in Agriculture or Healthcare, e.g., early stages of certain crop diseases or premature malignant tumors in humans.

Anomaly detection is mostly concerned with hard classification problems at early-stages of abnormalities in certain domains (i.e. crop, human diseases, chip manufacturing), which suffer from lack of data instances, and whose effective treatment would make a dramatic impact in these domains. For instance, fungus' visual cues on crops in agriculture or early-stage malignant tumors in the medical domain are hardly detectable in the relevant time-window, while the highly infectious nature

leads rapidly to devastation in a large scale. Other examples include detecting the faults in chip manufacturing industry, automated insulation defect detection with thermography data, assessments of installed solar capacity based on earth observation data, and nature reserve monitoring with remote sensing and deep learning. However, class imbalance poses an obstacle when addressing each of these applications.

In Precision Agriculture, and particularly in Precision Crop Protection [151], certain visual cues must be recognized with high accuracy in early stages of infectious diseases' development. A renowned use-case is the Potato Late Blight, with dramatic historical and economical impacts [152], whose early detection in field settings remains an open challenge (despite progress achieved in related learning tasks; see, e.g., [153]). The hard challenge stems from the actual nature of the visual cues (which resemble soil stains and are hardly distinguishable), but primarily from the fact that well-recording those early-stage indications is a rare event.

In the case of highly imbalanced datasets, such as those involving fraud or machine failure, it may be worth considering whether these examples can be classified as anomalies. If the problem meets the criteria for anomaly detection, techniques such as OneClassSVM, clustering methods, or Gaussian anomaly detection methods may be employed. These approaches involve considering the minority class as the outlier class, potentially providing new ways to classify and differentiate. Change detection is a similar concept to anomaly detection, but focuses on identifying changes or differences rather than anomalies. Examples of this include changes in user behavior as indicated by usage patterns or bank transactions.

In recent years, reliable models capable of learning from small samples have been obtained through various approaches, such as autoencoders [154], class-balanced loss (CBL) to find the effective number of samples required [155], fine tuning with transfer learning [156], data augmentation [75], cosine loss utilizing (replacing categorical cross entropy) [66], or prior knowledge [48].

3.6 Conclusion

As we conclude Chapter 3, our exploration through the diverse landscape of strategies and methodologies essential for effective learning from small datasets has been both comprehensive and insightful. This journey has armed us with a robust toolkit, encompassing a broad spectrum of techniques ranging from data augmentation, ensemble methods, and transfer learning to more intricate strategies like parameter initialization, loss function reformulation, and regularization techniques. We have also ventured into advanced realms such as synthetic data generation, physics-informed neural networks, and various forms of learning including unsupervised, semi-supervised, self-supervised, zero-shot, one-shot, few-shot, metalearning, and beyond. These techniques, together with insights into tackling specific challenges like imbalanced data and anomaly detection, form a crucial foundation for the focused applications and advanced methods explored in the subsequent chapters.

In Chapter 4 and 5, we build directly upon this foundation, delving into the nuances of learning from imbalanced datasets and exploiting the potential of self-supervised learning in scenarios of limited data. We have laid the groundwork for understanding how each of these advanced techniques and strategies can be effectively applied to real-world problems where data scarcity is a significant challenge. This chapter, therefore, acts as a pivotal link in our dissertation, ensuring a seamless and cohesive narrative that underscores the importance of mastering small data learning. Ultimately, the insights and methodologies discussed here contribute to the development of more robust and efficient machine learning models, capable of addressing the complex challenges presented by small datasets in various domains.

As we draw Chapter 3 to a close, we reflect on the extensive exploration of foundational concepts and methodologies critical for learning effectively from small datasets. The techniques and insights garnered here, from data augmentation to transfer learning and beyond, are not just theoretical constructs but essential tools that underpin the advanced topics in Chapters 4 and 5. The upcoming chapters build upon this bedrock, delving into the nuances of learning from imbalanced datasets (Chapter 4) and exploring the potential of self-supervised learning with salient image segmentation (Chapter 5), both of which hinge on the principles

established in this chapter. By connecting these concepts, we ensure a seamless transition into the intricacies of batch normalization and the dynamics of image segmentation in self-supervised learning. This chapter, thus, is a bridge that not only links but also enriches the narrative flow of the dissertation, ensuring that the journey from foundational principles to advanced applications in machine learning is both cohesive and comprehensive.