Developing UMAC: A Unified Model-Agnostic Computation Process for Enhanced Machine Learning Explainability

Elie Neghawi

A Thesis
In the Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Electrical and Computer Engineering)

at

Concordia University
Gina Cody School of Engineering and Computer Science
Department of Electrical and Computer Engineering
Montreal, Quebec, Canada

September 2024

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Elie Neghawi

Entitled: A Unified Model-Agnostic Computation Process for Enhanced Machine Learning Explainability

and submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY Electrical and Computer Engineering

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ **Chair**
Glenn Cowan

_____ **Thesis Supervisor**
Yan Liu

_____ **Examiner**
Dongyu Qiu

_____ **Examiner**
Anjali Agarwal

_____ **External Examiner**
Tse-Hsun (Peter) Chen

_____ **Arms-Length Examiner**
Martin Litoiu

Approved by_____
Chair of Department or Graduate Program Director

Defence Date: 01/14/2025 _____
Dean, *Mourad Debbabi*

Abstract

Developing UMAC: A Unified Model-Agnostic Computation Process for Enhanced Machine Learning Explainability

Elie Neghawi, Ph.D.

Concordia University, 2024

The rapid evolution of Convolutional Neural Networks (CNNs) has produced increasingly efficient and versatile algorithms, but the factors driving their superior performance remain underexplored. While previous research has primarily focused on explaining Semi-Supervised Machine Learning (SSML) algorithms in a model-specific manner, this thesis aims to generalize those findings, making them applicable across a wider range of CNNs. The challenge lies in achieving a method that can both enhance performance and improve interpretability, while remaining adaptable to various models.

This thesis introduces a post-hoc Explainable Artificial Intelligence (XAI) method, called Unified Model Agnostic Computation (Unified Model-Agnostic Computation (UMAC)), designed to generalize common components of CNNs models by drawing insights from SSML and Self-Supervised Learning (SSL) algorithms. Our research begins by focusing on two primary aspects: (1) the effect of parameter updates during training on both labeled and unlabeled data in SSML and SSL, and (2) the transition from model-specific SSML frameworks to a more generalized, model-agnostic approach using SSL.

In the first phase, we used SSMLs as a foundation, breaking down their components into preprocess-centric and classifier-centric elements, which led to the creation of Semi-Supervised Computation Processs (SSCPs) (Semi-Supervised Computation Processes). These processes were tested across five state-of-the-art SSML algorithms and three SSL algorithms, using various Deep Neural Networks (DNNs). Although this phase acted as a testing ground to understand the mechanics of SSML, it allowed us to identify key drivers of performance, especially in relation to parameter updates and data handling.

Through 45 rigorous experiments, we observed an 8% reduction in training loss and a 6.75% increase in learning precision using the Shake-Shake26 classifier with the RemixMatch SSML algorithm. A key observation was the positive correlation between labeled data and training time, showcasing the importance of label quantity in enhancing model efficiency.

In the second phase, we transitioned from SSML to SSL to prove that the methodology could be generalized to a model-agnostic approach. By integrating SSL components, we aimed to develop a unified framework that worked across various DNNs architectures. Building upon this analysis, we developed a UMAC process for SSL, tailored to complement modern self-supervised learning algorithms. UMAC serves as a model-agnostic XAI methodology that explains models by composition, systematically integrating and enhancing state-of-the-art algorithms. Through UMAC, we identified key computational mechanisms and crafted a unified framework for self-supervised learning evaluation. Our systematic approach yielded a 17.12% improvement in training time complexity and a 13.1% boost in testing time complexity, with notable improvements observed in augmentation, encoder architecture, and auxiliary components within the network classifier. This phase demonstrated that UMAC could enhance accuracy and reduce training loss under different data conditions, showing its adaptability to different models and datasets.

In the third phase, we applied the UMAC framework to the field of medical image classification. Medical imaging tasks often suffer from data scarcity, making it challenging to achieve both high performance and model interpretability. By leveraging the UMAC methodology, we integrated it into CNNs and Transformers to generate high-quality representations, even with limited data. Experiments across five 2D medical image datasets showed that UMAC outperformed traditional augmentation methods by 1.89% in classification accuracy. Additionally, incorporating explainable AI (XAI) techniques ensured that the models provided transparent and reliable decision-making processes, enhancing their interpretability in critical medical applications.

Throughout this process, UMAC served as an XAI method based on explaining models by composition, systematically breaking down computational processes to reveal how model components contribute to overall performance. This approach enabled us to create a unified, model-agnostic framework that enhanced both transparency and efficiency in CNNs.

Ultimately, this thesis contributes a structured and generalizable approach for Machine Learnings (MLs) developers, offering step-by-step guidelines to improve model performance and interpretability. By generalizing the computation processes of SSML and SSL through the UMAC framework, we provide developers with the tools needed to optimize their models across various domains, particularly in fields where transparency and accuracy are critical.

Contents

List of Figures

List of Tables

Abbreviations

Abbreviation	Description
ACC	Accuracy
AI	Artificial Intelligence
AUC	Area Under the ROC Curve
BSDA	Bayesian Semantic Data Augmentation
BYOL	Bootstrap Your Own Latent
CF-SSCP	Classifer Focused Semi-Supervised Computation Process
CNN	Convolutional Neural Network
DNN	Deep Neural Network
EMA	Exponential Moving Average
GNN	Graph Neural Network
ISDA	Implicit Semantic Data Augmentation
ML	Machine Learning
MLP	Multi-layer Perceptron
MoCo	Momentum Contrast
MoCov2	Momentum Contrast Version 2
PF-SSCP	Preprocessing Focused Semi-Supervised Computation Process
SGD	Stochastic Gradient Descent
SimCLR	Simple Contrastive Learning of Representations Version
SimCLRv2	Simple Contrastive Learning of Representations Version 2
SOTA	State Of The Art
SSCP	Semi-Supervised Computation Process
SSL	Self-Supervised Learning
SSML	Semi-Supervised Machine Learning
UMAC	Unified Model-Agnostic Computation
ViT	Vision Transformer
XAI	Explainable Artificial Intelligence

Table 1: Abbreviations

1 Introduction

ML algorithms are increasingly recognized for their capacity to solve complex tasks across various domains, yet they often face significant challenges due to the scarcity of labeled data. Collecting and annotating large amounts of labeled data can be both resource-intensive and time-consuming, particularly in specialized fields such as medical imaging, autonomous systems, and scientific research. This lack of labeled data hampers the performance of traditional machine learning systems, which rely heavily on supervised learning methods that require vast labeled datasets for optimal performance.

To mitigate this challenge, researchers have increasingly turned to semi-supervised learning approaches, which combine labeled data with the vast amounts of unlabeled data typically available. Semi-supervised machine learning (SSML) algorithms work by coupling a network classifier with two primary components: a classification cost, which is computed using labeled data, and a consistency cost, which captures the differences between augmentations of unlabeled data processed by the classifier. These algorithms, such as the Π-model, Temporal Ensembling, and Mean-Teacher, have demonstrated success in leveraging unlabeled data to improve learning efficiency. Moreover, deep neural networks (DNNs) have further enhanced the potential of SSMLs by introducing architectures that allow for more sophisticated pattern recognition and feature extraction.

Despite these advancements, the underlying mechanisms that contribute to the superior performance of SSMLs models when integrated with DNNs remain poorly understood. Current research often treats the network classifier as a black box, which limits our understanding of the key factors influencing training performance, including training loss, learning accuracy, and model optimization. This lack of transparency leads to ad-hoc and model-specific tuning, which does not provide a scalable, systematic way to generalize SSMLs across different models and datasets.

The motivation for this thesis is to address a critical gap in the interpretability and optimization of machine learning models by developing a comprehensive framework that enhances performance while providing explainability through an XAI lens. This thesis introduces a post-hoc XAI method, called Unified Model-Agnostic Computation (UMAC), designed to generalize common components of CNNs models by drawing insights from both semi-supervised machine learning (SSML) and self-supervised learning (SSL) algorithms. Unlike model-specific approaches, UMAC adopts

a model-agnostic perspective, enabling it to generalize across different architectures and learning paradigms. Our research begins by focusing on two primary aspects: (1) the effect of parameter updates during training on both labeled and unlabeled data in SSMLs and SSLs, and (2) the transition from model-specific SSMLs frameworks to a more generalized, model-agnostic approach using SSLs. By visualizing the internal processes of these models, UMAC provides a powerful tool for both understanding and optimizing complex CNNs and learning models, ultimately contributing to improved model performance and transparency across various machine learning applications.

In this work, we present a systematic approach to identifying the key factors that impact the training and learning accuracy of SSMLs when used with DNNs. By doing so, we aim to provide concrete insights that will allow machine learning developers to optimize models more effectively, contributing both to academic research and practical applications.

In the first phase of this research, we introduced SSCPs (Semi-Supervised Computation Processes) for SSML, categorizing them into preprocess-centric and classifier-centric components. These SSCPs facilitated the modular composition of SSMLs algorithms. Through extensive experiments involving five state-of-the-art SSMLs and three SSLs algorithms, applied across three DNNs at varying labeled/unlabeled data ratios, we identified and analyzed their peak performances. The findings, presented as computation graphs, highlight peak performance metrics and parameter update processes for each algorithm.

This stage of the research was directed by the following key question:

RQ1: What are the most influential factors impacting computational costs and training accuracy when integrating deep neural networks (DNN) with semi-supervised machine learning (SSML)?

This primary research question (**RQ1**) is further explored through two sub-questions:

- **RQ1.1**: How do variations in parameter updates and preprocessing techniques affect training time, learning accuracy, and training loss in network classifiers within SSML models? (Answer in Section 5.3)
- **RQ1.2**: How do different methods for computing consistency and classification costs influence the effectiveness of SSML models in terms of accuracy and loss during the training and inference phases? (Answer in Section 5.3)

In the second phase, we aimed to generalize our methodology to create a model-agnostic approach. Building upon our previous analysis, we developed a UMAC process for SSLs, tailored to complement modern SSLs algorithms. UMAC serves as a model-agnostic XAI methodology that explains models by composition, systematically integrating and enhancing state-of-the-art algorithms. Through UMAC, we identified key computational mechanisms and crafted a unified framework for SSLs evaluation. Our systematic approach yielded notable improvements in training and testing time complexity, with significant enhancements observed in augmentation, encoder architecture, and auxiliary components within the network classifier.

This stage of the research was guided by the following inquiries:

- **RQ2**: What are the key components necessary to define a unified computational process for evaluating SSLs algorithms? (Answer in Section 6.5.1)
- **RQ3**: How can the unified computational process be tailored to improve the time complexity and interpretability of SSLs algorithms?(Answer in Section 6.5.1)

In the third phase of this research, we extend the application of the UMAC framework to the field of medical image classification. While deep learning methods have significantly advanced diagnostic accuracy in clinical settings, they often require large datasets, which are scarce in medical contexts [Litjens et al., 2017]. Common methods like data augmentation and generative models, though helpful, are computationally expensive and often insufficient in improving sample diversity without introducing biases [Shorten and Khoshgoftaar, 2019, Cubuk et al., 2019, Perez and Wang, 2017, Ratner et al., 2017]. To address these challenges, recent feature-level data augmentation techniques, such as ISDA, have emerged as promising alternatives [Wang et al., 2019]. However,

these methods still lack the balance between semantic direction and strength.

Building on our previous work [Neghawi and Liu, 2024], we apply UMAC to tackle these issues in the medical field, where data scarcity, model generalization, and interpretability are critical. UMAC, with its model-agnostic and adaptable design, integrates well into medical applications, helping to improve performance across different modalities and architectures, such as CNNs and Transformers. By generating new models that are tailored for medical image classification and incorporating XAI methodologies, UMAC ensures more effective and transparent machine learning solutions.

By generalizing the computation processes of SSMLs and SSLs through the UMAC framework, our XAI method provides a model-agnostic approach that enhances the interpretability and efficiency of CNNs models. This thesis contributes to the understanding of the underlying mechanisms of advanced machine learning algorithms and offers strategic insights for ML developers and engineers to optimize their models for better performance and transparency.

Publications:

Our research has been reviewed and accepted by various venues, reflecting recognition from the academic community. The key publications resulting from this work, along with their relevant sections, are as follows:

- Neghawi, E., Liu, Y. (2020). "Evaluation of Parameter Update Effects in Deep Semi-Supervised Learning Algorithms," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 351-360 [Neghawi and Liu, 2020]. [Impact Factor: 1.66] — This publication is central to the discussion in Sections 4.3 and 5.

- Neghawi, E., Liu, Y. (2023). "Analysing Semi-Supervised ConvNet Model Performance with Computation Processes," *Machine Learning and Knowledge Extraction*, 5(4), 1848-1876 [Neghawi and Liu, 2023]. [Impact Factor: 3.44] — Relevant to the analysis in Sections 4.3 and 5.

- Neghawi, E., Liu, Y. (2024). "Enhancing Self-Supervised Learning through Explainable Artificial Intelligence Mechanisms: A Computational Analysis," *Big Data and Cognitive Computing*, 8(6), Article 58 [Neghawi and Liu, 2024]. [Impact Factor: 2.51] — Discussed

in Section 6 and the transition to UMAC.

- Neghawi, E., Wang, Z., Huang, J., Liu, Y. (2023). "Linking Team-level and Organization-level Governance in Machine Learning Operations through Explainable AI and Responsible AI Connector," *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 1223-1230 [Neghawi et al., 2023]. [Impact Factor: 1.66] — Discussed in Section 7.

2 Related work

Several approaches exist for explaining CNNs, with one of the most notable being the unification and taxonomic view of Graph Neural Networks (GNNs) [Baldassarre and Azizpour, 2019]. This significant work delineates the commonalities and distinctions among existing methodologies, offering a foundation for future advancements [Yuan et al., 2020b]. The evaluation is streamlined by creating a suite of benchmark graph datasets specifically for GNN explainability. This encompasses metrics and datasets essential for appraising GNN explainability [Xie et al., 2020]. Nonetheless, complexities escalate when these concepts are applied to SSML algorithms due to the employment of multiple DNNs as network classifiers. Our approach delves into model-level explanations, directing attention to the taxonomy of SSMLs, and underscores precise changes and their driving forces.

An alternative study presented an interpretable compositional CNN approach [Shen et al., 2021, Doshi-Velez and Kim, 2017], aiming to refine the conventional convolutional neural network (CNN) structure. This strategy is designed to facilitate the learning of filters that capture meaningful visual motifs within intermediate convolutional strata. While effective for illustrating a singular CNN, its applicability falters with SSML, where the network classifier integrates two or more CNNs.

Additionally, the XGNN method, which is a novel approach to graph generation, trains a graph generator to approximate predictions for target graphs rather than optimizing the input graph directly [Yuan et al., 2020a]. The graphs generated by this method are considered to be accurate explanatory targets that capture the unique patterns characteristic of graphs. The XGNN's graph generation process uses reinforcement learning, allowing it to incorporate any competent graph

generation algorithm within its framework, which broadens the scope and applicability of its explanations. This approach enhances the overall understanding of how trained graph neural networks (GNNs) function. However, while XGNN demonstrates strong performance in explaining graph classification models, its application to SSMLs algorithms is less effective. To address this, our technique leverages the robust graph generation capabilities of XGNN but enhances them by adding a layer of decomposition based on instance-level explanations [Pope et al., 2019]. This serves to shed light on three critical aspects of the latest SSMLs algorithms, providing a more nuanced understanding.

Recent advancements in SSMLs extensively use certain techniques, including consistency regularization, self-training, and entropy minimization. The study reported in [Kim, 2021, Tarvainen and Valpola, 2017c] indicates that SSMLs methods may experience a significant decline in performance when there is a discrepancy between the distribution of unlabeled data used during training and those encountered post-training, even if other variables such as initialization, preprocessing, regularization, and augmentation are well-managed. Conversely, our research focuses on developing a generalized approach to these techniques and elucidating the specific procedural steps that lead to better performance, independent of the specific characteristics of the unlabeled data.

Informed machine learning [Kim, 2021] integrates supplementary prior knowledge during model training. The referenced study provides a comprehensive synopsis of diverse strategies within this domain, meticulously outlining the informed machine learning design process and its numerous constituent elements. It also clearly distinguishes informed ML from conventional ML paradigms. Our research, however, zeroes in on SSMLs algorithms, starting with the initial design, then delving deeper into the taxonomy of each and revealing the influential factors in every segment.

Self-supervised learning is positioning itself as a cornerstone in the machine learning landscape. At its core, this paradigm exploits the structure of unlabeled data to extract meaningful representations through intricate pretext tasks. The survey by [Jaiswal et al., 2021] offers valuable insights into this domain. However, it predominantly delves into individual algorithmic accomplishments, missing out on a holistic perspective that encompasses the shared attributes and foundational components uniting various self-supervised approaches.

Parallel to the advancements in self-supervised learning, the domain of XAI has garnered

considerable attention [Rudin, 2019, Baldassarre and Azizpour, 2019]. Methods such as LIME [Ribeiro et al., 2016] and SHAP [Lundberg and Lee, 2017] have emerged as frontrunners in offering model interpretability. Nevertheless, it is paramount to note that the XAI technique proposed in our research diverges from these methodologies, both in conception and execution [Xie et al., 2020].

Another study introduces the Contrastive Learning with Stronger Augmentations (CLSA) framework [AuthorLastName and CoAuthorLastName, Year]. This approach capitalizes on the potential of stronger augmentations in contrastive learning. However, while the work elucidates the benefits of their method, it falls short of thoroughly explaining the underlying mechanisms driving its performance gains.

2.1 XAI Related Topics

In the study "XAI for Self-supervised Clustering of Wireless Spectrum Activity" [Milosheski and et al., 2023], the authors delve into the interpretability challenges of deep learning models, particularly in the wireless communications sphere. Their method integrates CNN-based representation learning with deep clustering tailored for wireless spectrum activity. Although their work offers valuable insights in its specific domain, it is primarily application-centric. On the other hand, our research extends a comprehensive exploration into diverse self-supervised learning architectures and the breakthroughs that they bring.

Shifting the focus to the medical imaging realm, recent research has probed into the effectiveness of self-supervised representation learning using fetal ultrasound videos from mid-pregnancy [Jiao et al., 2020]. In this context, explainability is equated with capturing anatomy-aware knowledge. A set of quantitative metrics, anchored on visually salient landmarks, is introduced [Droste et al., 2019]. By honing in on the quality of landmark CNN feature clustering, the study suggests that such features hold the key to understanding anatomy-aware insights. These metrics not only guide the choice of an apt self-supervised learning method without delving into downstream tasks but also ensure that Artificial Intelligence (AI) explanations resonate with clinical significance. However, it is important to note that this approach may have inherent limitations: its focus is specifically on fetal ultrasound imaging, potentially limiting its generalizability to other medical

imaging modalities or broader applications beyond the realm of medical imaging.

2.2 Architecture Patterns in the Software Domain

In the software engineering domain, architecture patterns play a crucial role in enabling modularity, flexibility, and reusability in system design. One of the most significant aspects of these patterns is their ability to provide a ”plug-and-play” framework, where components can be independently developed, tested, and integrated without requiring extensive modifications to the existing system [Fowler, 2018]. This approach allows for a higher degree of flexibility, as individual components can be easily swapped or upgraded to enhance functionality without disrupting the overall system architecture. Examples of such architecture patterns include microservices [Newman, 2015] and event-driven architectures [Richards, 2015], both of which are widely adopted in modern software systems for their ability to decouple components and enable scalability.

In the context of machine learning frameworks like UMAC, adopting architecture patterns from the software domain can offer similar benefits. By decomposing the system into modular, interchangeable components, it becomes possible to integrate different data processing pipelines, models, and augmentation techniques without needing to redesign the entire system. This ”plug-and-play” characteristic of software architecture patterns is instrumental in enhancing the adaptability and scalability of machine learning solutions across various domains, particularly in handling diverse data modalities and architectures such as those in medical imaging.

2.3 Process Decomposition in Other Fields

While the concept of architectural modularity is well-explored in software engineering, other fields also employ decomposition techniques to break down complex processes into manageable components. In fields such as physics and systems biology, decomposition often focuses on refining processes at different scales, where refinement strategies aim to detail the system’s behavior at varying levels of abstraction [Gillies, 2012, Kitano, 2002]. This process of decomposition enables researchers to progressively refine models to better capture the intricacies of complex systems. However, unlike in the software domain, where decomposition is frequently discussed in terms of modularity and plug-and-play adaptability, these fields typically approach decomposition as a

means of achieving greater precision and refinement, with less emphasis on modularity and interchangeability.

In the machine learning domain, and particularly within the context of UMAC, there has been limited discussion regarding process decomposition beyond the refinement of individual components [Li et al., 2020]. While some work has focused on improving model accuracy through iterative refinements of training processes and data augmentation methods, the broader application of decomposition as a strategy for modular and interchangeable processes, akin to architecture patterns in software, remains underexplored. This presents an opportunity for future research to examine how decomposition techniques from other scientific disciplines can be adapted to machine learning frameworks like UMAC, particularly in enhancing the modularity and flexibility of machine learning models across various domains.

3 The Background

In the dynamic landscape of machine learning, semi-supervised learning (SSML) and self-supervised learning (SSL) are two pivotal methodologies, each contributing unique strategies and computational models. This section provides a succinct yet comprehensive analysis, beginning with SSML and then transitioning to SSL.

We start with an exploration of SSML, examining state-of-the-art models like Temporal Ensembling, the Π-Model, Mean Teacher, MixMatch and ReMixMatch. Focusing on a modular-based analysis, this exploration aims to distill the core computational components of these models, highlighting the functional attributes that are crucial for enhancing learning performance in SSML scenarios.

After discussing SSML, we shift our attention to SSL. Here, we delve into advanced SSL . These methods exemplify how SSL leverages unlabeled data, autonomously generating supervisory signals to facilitate learning. This part of the analysis focuses on the foundational principles and innovations within SSL, illustrating how these approaches are shaping the future of autonomous and efficient learning methodologies.

Overall, this section bridges theoretical insights with practical applications in both SSML and SSL. By providing an integrated overview of these methodologies, we aim to illuminate their syner-

gies and individual impacts on the broader field of machine learning, underscoring their importance in driving future advancements and innovations in the domain.

3.1 The Analysis of SSML Models

This subsection introduces the core SSML models, each contributing a distinct methodology for handling the balance between labeled and unlabeled data. These models—Temporal Ensembling, the Π-Model, Mean Teacher, MixMatch, and ReMixMatch—form the foundation for many modern SSML frameworks. Below, we provide a brief overview of each model, outlining its primary approach and significance within the SSML landscape.

3.1.1 Temporal Ensembling

Temporal Ensembling, introduced by [Laine and Aila, 2016], builds on the basic structure of semi-supervised learning (SSML) by using a single DNN as the network classifier. Unlike traditional models, which may use multiple networks or evaluate network augmentations multiple times per batch, Temporal Ensembling evaluates the network only once per epoch. The key innovation lies in its use of past predictions to update the consistency cost in subsequent epochs, making it both efficient and effective for training.

In this method, each data point x_i is processed by the DNN to produce a prediction z_i. These predictions are then aggregated into an ensemble prediction Z_i over time, with each new prediction being combined with the previous ensemble using a momentum term, α. This creates a smoother target for training, reducing the noise compared to models like the Π-model. To further enhance accuracy, a bias correction is applied, dividing the ensemble by $(1 - \alpha^t)$ to compensate for early inaccuracies during training.

The main advantages of Temporal Ensembling include:

- Faster training by reducing the number of evaluations per input to once per epoch.
- Improved training stability and accuracy through reduced noise in the ensemble targets.

However, it also has some limitations:

- Requires additional storage for ensemble predictions and the momentum parameter α.

- Does not support dynamic learning, requiring retraining of the model to incorporate new data effectively.

In summary, Temporal Ensembling optimizes training efficiency and reduces noise by leveraging past predictions, making it suitable for scenarios where computational resources are constrained, yet high model accuracy is essential. It operates similarly to other SSML methods, balancing classification cost and consistency cost, but relies on accumulated ensemble predictions to compute consistency loss.

3.1.2 Π-model

The Π-model [Laine and Aila, 2016], a variant of SSML, implements a self-ensembling mechanism where consistency is enforced between two versions of the same input, processed under different dropout conditions and augmentations [Laine and Aila, 2016]. This method helps the network achieve better generalization by ensuring that the outputs for slightly perturbed inputs remain consistent.

Both labeled and unlabeled data x_i undergo identical preprocessing and are passed through two identical neural networks, differentiated by the randomness introduced through dropout and data augmentations. These networks produce predictions z_i and $\tilde{z}_i$, which are used to calculate the total loss. For labeled data, both Classification and Consistency Costs are computed, while for unlabeled data, only the Consistency Cost is considered. The weighted sum of these losses, controlled by a ramp-up function $w(t)$, adjusts the importance of the Consistency Cost during training, prioritizing Classification Cost in the early stages. The key advantages of the Π-model include:

- Improved accuracy compared to earlier SSML methods by promoting consistency between outputs.
- Ability to regularize network predictions through dropout and augmentations.

However, the Π-model has several limitations:

- Increased computational cost, as the network output is calculated twice for each input.
- Sensitivity to incorrectly labeled data, which can degrade the model's performance.
- Inability to support dynamic or online learning, as it relies on batch learning.

3.1.3 Mean Teacher

The Mean Teacher model, introduced by [Laine and Aila, 2016], is an extension of the SSML family that addresses the limitations of the Π-model and Temporal Ensembling by averaging the weights of consecutive models rather than directly using the final prediction. The method involves two identical DNNs—referred to as the Student and Teacher models. The Teacher model's weights are updated as an Exponential Moving Average (EMA) of the Student model's weights, as described by Equation 2. This weight-averaging process reduces noise and improves model stability over time.

The Mean Teacher model processes data similarly to the Π-model during forward propagation for both labeled and unlabeled data. However, in backward propagation, only the Student model's parameters are updated through gradient descent. The Teacher model's parameters are updated via EMA, ensuring smoother learning and more stable convergence.

The key advantages of the Mean Teacher model include:

- More accurate model performance due to the averaging of weights over multiple training steps.
- Faster training time compared to the Π-model, while achieving better accuracy than Temporal Ensembling.

However, there are some trade-offs:

- Increased memory usage, as both models (Student and Teacher) must maintain different dropout conditions and parameters.
- Only the Student model undergoes backward propagation, meaning the Teacher model does not directly contribute to gradient updates.

Overall, the Mean Teacher model provides a more stable learning process by smoothing the parameter updates over time, offering a balance between accuracy and training efficiency in semi-supervised learning.

3.1.4 MixMatch

MixMatch, unlike previous SSML models, employs a different strategy by augmenting both labeled and unlabeled data during each training batch [Berthelot et al., 2019]. For labeled data, the raw inputs x_b are augmented, keeping the labels y_b, and generating a dataset $\hat{X}$. For unlabeled data, the inputs are augmented K times (where K is a hyperparameter), and predictions are averaged across the K augmentations to produce a final label prediction $\tilde{z}_{bavg}$. This prediction is then sharpened using temperature sharpening to reduce the entropy of the label distribution, making the predictions more confident.

MixMatch also introduces MixUp, a data augmentation technique applied to both labeled and unlabeled data. This process combines two samples (x_1, p_1) and (x_2, p_2) with a mixing coefficient λ, which is drawn from a Beta distribution. The mixed inputs and their corresponding labels are used for both labeled data (X') and pseudo-labeled unlabeled data (U'). This technique encourages smooth transitions between labeled and unlabeled data points, improving generalization.

The total loss function in MixMatch consists of two components:

- **Classification Loss** (L_X): A standard cross-entropy loss applied to the labeled data in the batch.
- **Consistency Loss** (L_U): An L_2 loss applied to the pseudo-labeled data, ensuring that the network's predictions for unlabeled data are consistent with the guessed labels.

The final loss is a weighted sum of these two components, with a weight λ_U applied to the consistency loss.

Key benefits of MixMatch include:

- Improved accuracy through label sharpening and mixing of labeled and unlabeled data.
- Effective use of unlabeled data by averaging predictions and applying pseudo-labeling techniques.

However, MixMatch does have certain limitations:

- The process involves several hyperparameters that need careful tuning, such as the number of augmentations (K) and the sharpening temperature (T).

- The model is sensitive to the quality of pseudo-labels, which could impact performance if the labels are inaccurate.

3.1.5 ReMixMatch

ReMixMatch builds upon the MixMatch framework by introducing several enhancements to improve accuracy and consistency in semi-supervised learning [Berthelot et al., 2020]. One key modification is the use of augmentation anchoring, where the labeled data undergoes strong augmentations through CTAugment, a variant of AutoAugment that learns an augmentation policy during training. The unlabeled data is augmented multiple times, generating both weakly and strongly augmented versions for use in training.

A second major enhancement in ReMixMatch is the implementation of distribution alignment. The model maintains a running average of its predictions for the unlabeled data and aligns them with the marginal class distribution from the labeled data. This alignment ensures that the network's predictions are consistent with the overall class distribution, which helps to mitigate the bias introduced by incorrect or skewed pseudo-labels.

ReMixMatch then applies temperature sharpening to further reduce the entropy of the pseudo-labels for the unlabeled data. The final dataset consists of both strongly augmented labeled and unlabeled data, which are combined and processed similarly to MixMatch, with MixUp used to blend the examples for more effective generalization.

Key improvements introduced in ReMixMatch:

- Augmentation anchoring with strong augmentations improves robustness in the model's predictions.
- Distribution alignment ensures a more balanced and accurate prediction set by aligning predictions with the marginal class distribution.
- Temperature sharpening reduces label entropy, making pseudo-labels more confident.

However, these modifications also bring new challenges:

- ReMixMatch requires more computational resources due to the multiple augmentations and the distribution alignment calculations.

- The effectiveness of the method depends on the quality of the augmentations and the learned augmentation policies, which may require fine-tuning.

In summary, ReMixMatch improves upon MixMatch by introducing stronger augmentations, distribution alignment, and refined pseudo-label sharpening, making it a more robust method for handling unlabeled data in semi-supervised learning.

3.1.6 Comparison of SSML Models

The following table 2 provides a summary of the key characteristics, advantages, and limitations of each SSML model discussed in the previous subsections:

Common Ground between Models:

- All models focus on utilizing unlabeled data to improve the learning process and enhance the accuracy of the network.
- Most of the models implement some form of consistency cost (Temporal Ensembling, Π-model, Mean Teacher, MixMatch, ReMixMatch), ensuring stable learning from both labeled and unlabeled data.
- Many methods (Temporal Ensembling, Π-model, Mean Teacher) rely on ensemble techniques or weight averaging to improve prediction accuracy and reduce noise during training.

Key Differences:

- **Consistency Enforcement**: Temporal Ensembling relies on predictions from past epochs, while the Π-model and Mean Teacher ensure consistency within a single batch through dropout/augmentation or weight averaging.
- **Augmentation Techniques**: MixMatch and ReMixMatch use advanced augmentation techniques (e.g., MixUp and CTAugment) and label sharpening, which distinguishes them from the earlier models.
- **Training Efficiency**: Temporal Ensembling evaluates predictions once per epoch, offering a faster training time compared to the Π-model and MixMatch, which evaluate multiple versions of the input.

Model	Key Characteristics	Advantages	Limitations
Temporal Ensembling	Uses past predictions to update consistency cost over epochs. Evaluates network once per epoch.	Faster training, smoother training targets with reduced noise.	Requires additional storage for predictions and momentum term. Does not support dynamic learning.
Π-model	Enforces consistency between predictions from two identical networks with different augmentations and dropout conditions.	Improved generalization, regularizes network predictions through augmentations.	Increased computational cost, sensitive to mislabeled data, not suitable for dynamic learning.
Mean Teacher	Uses an EMA of the weights of the Student network for the Teacher network.	More stable learning through weight averaging, faster training time than the Π-model.	Increased memory usage, Teacher model does not contribute to gradient updates directly.
MixMatch	Augments both labeled and unlabeled data, averages multiple predictions, sharpens labels, and uses MixUp for smooth transitions.	Improved accuracy through label sharpening and mixing of labeled/unlabeled data.	Sensitive to pseudo-label quality, requires careful tuning of hyperparameters such as temperature and augmentations.
ReMixMatch	Enhances MixMatch with stronger augmentations (CTAugment), distribution alignment, and more refined sharpening.	Robust predictions through augmentation anchoring, distribution alignment, and improved label confidence.	Requires more computational resources, relies on the quality of learned augmentation policies and tuning.

Table 2: Summary of SSML models

- **Complexity and Resources**: ReMixMatch, with its augmentation anchoring and distribution alignment, requires more computational resources, but it also provides more robust and accurate predictions compared to simpler methods like the Π-model.

3.2 The Analysis of SSL Models

SSL models have pushed boundaries in machine learning, particularly when labels are scarce or expensive to obtain. Each model we discuss takes a unique approach to contrastive learning, and understanding their differences provides insight into how SSL has evolved. Below, we explore

the nuances and contributions of several key models, each with its own set of innovations and challenges.

3.2.1 MoCo (Momentum Contrast)

MoCo [He et al., 2020] tackles a common problem in contrastive learning: maintaining a reliable and large pool of negative samples. Rather than depending solely on the current mini-batch for negatives, MoCo introduces a dynamic dictionary that decouples this process. This dictionary allows the model to store representations from previous mini-batches, creating a larger set of negatives for contrastive learning.

The standout feature of MoCo is its momentum encoder, which smooths the updating of dictionary parameters. Instead of updating after every mini-batch, it applies a momentum-based update:

$$\theta_t = m\theta_{t-1} + (1-m)\theta_{\text{main}}$$

This helps maintain consistency in the representation of negative samples over time, avoiding drastic shifts that can hinder learning.

Key benefits:

- A large and stable pool of negative samples improves contrastive learning performance.
- The momentum-based updates ensure more consistent representations, stabilizing the learning process.
- MoCo scales well to large datasets due to its dynamic dictionary mechanism.

3.2.2 MoCov2 (Momentum Contrast v2)

Building on MoCo, MoCov2 [Chen et al., 2020e] refines key components to push performance further. The MLP head introduced in MoCov2 is crucial, as it allows for better feature mapping in the latent space, leading to improved contrastive learning.

Another upgrade is the adoption of a stronger augmentation strategy, such as more aggressive color distortions, to further increase data diversity during training. Additionally, MoCov2

introduces cosine annealing for learning rates, which provides smoother learning rate transitions, preventing abrupt changes that can destabilize training.

What sets MoCov2 apart:

- The enhanced MLP head helps in learning better feature representations.
- Stronger augmentations increase robustness and generalization.
- Cosine annealing ensures that learning progresses smoothly without oscillations.

3.2.3 SimCLR (Simple Framework for Contrastive Learning)

SimCLR [Chen et al., 2020b] simplifies contrastive learning by eliminating complex components like momentum encoders or dynamic dictionaries. Instead, SimCLR relies heavily on data augmentation to achieve effective learning. The idea is simple: augment the same image in multiple ways and then train the model to maximize the similarity between the two views while minimizing similarity to other images in the dataset.

To make this work, SimCLR employs large batch sizes to ensure that enough negative samples are present in each batch. A combination of augmentations like cropping, color jitter, and Gaussian blur is used to introduce variability in the input data, making the model more robust to changes in the input distribution.

Advantages:

- SimCLR's heavy reliance on augmentations allows it to learn powerful representations without the need for a separate encoder or dictionary.
- Its simplicity makes it easy to implement and scale, particularly on large datasets.
- The large batch size ensures there are enough negatives for effective contrastive learning.

3.2.4 SimCLRv2 (Simple Framework for Contrastive Learning v2)

SimCLRv2 [Chen et al., 2020c] takes SimCLR's strengths and builds on them by adding a non-linear projection head. This extra layer allows the model to better separate features in the latent space, improving the quality of learned representations.

SimCLRv2 also uses larger encoders, such as ResNet-50 and ResNet-101, to increase the capacity of the model. After self-supervised pretraining, SimCLRv2 incorporates a supervised fine-tuning step, allowing the model to benefit from labeled data to further refine its representations.

What's new in SimCLRv2:

- The non-linear projection head improves feature separation, leading to better downstream performance.
- Larger encoders allow the model to capture more complex patterns in the data.
- The supervised fine-tuning step leverages labeled data to refine the representations, boosting performance on labeled tasks.

3.2.5 BYOL (Bootstrap Your Own Latent)

BYOL [Grill et al., 2020] breaks from traditional contrastive learning by removing the need for negative samples entirely. Instead, BYOL focuses on learning good representations by aligning two augmented views of the same image, using an online network and a target network.

The target network is an exponential moving average of the online network, meaning it is updated more slowly to provide a consistent target for the online network to match. This setup prevents the representations from collapsing to trivial solutions (e.g., all representations becoming identical), a common risk when negative samples are absent.

Why BYOL is different:

- No need for negative samples, simplifying the training process.
- The target network provides stable targets, reducing the risk of representation collapse.
- BYOL achieves high performance without the need for contrastive pairs, challenging the assumption that negatives are necessary for SSL.

3.2.6 Comparison of SSL Models

Table 3 provides a comparison of key SSL models, summarizing their unique characteristics, advantages, and limitations. It highlights how models like MoCo and SimCLR rely on negative samples, while BYOL removes this requirement entirely. Additionally, models such as MoCov2 and

SimCLRv2 improve performance through stronger augmentations and larger architectures. Each model addresses different challenges in SSL, making this table a quick reference for understanding their distinct approaches.

Model	Key Characteristics	Advantages	Limitations
MoCo	Introduces a dynamic dictionary with a momentum encoder to store and update negative samples from previous batches.	Large, stable pool of negatives; momentum-based updates improve stability.	Requires additional memory for dictionary storage; requires momentum parameter tuning.
MoCov2	Builds on MoCo with an MLP head and stronger augmentations, as well as cosine annealing for the learning rate.	Improved feature mapping and robustness through augmentations and smooth learning rate transitions.	Computationally intensive due to stronger augmentations and larger models.
SimCLR	Simplifies contrastive learning, relying on augmentations and large batch sizes to provide sufficient negatives.	Easy to implement and scale, effective use of augmentations.	Requires very large batch sizes for sufficient negative samples, computational cost increases with batch size.
SimCLRv2	Extends SimCLR with a non-linear projection head and larger encoders, plus supervised fine-tuning after self-supervised pretraining.	Better feature separation, larger model capacity, fine-tuning improves downstream performance.	Larger encoders and fine-tuning add to the computational cost.
BYOL	Eliminates the need for negative samples, uses an exponential moving average target network for stable learning.	No need for negatives, stable targets prevent representation collapse, high performance without contrastive pairs.	Risk of trivial solutions if improperly configured; depends on carefully balancing the target and online networks.

Table 3: Comparison of SSL Models

Common Ground between Models:

- All models focus on learning meaningful representations from unlabeled data, a core tenet of self-supervised learning (SSL).

- Most models utilize data augmentation in some form to create different views of the input data, helping the model learn robust representations.
- Many of the models aim to stabilize learning either through momentum-based updates (MoCo, BYOL) or large batch sizes (SimCLR).

Key Differences:

- **Negative Samples**: MoCo and SimCLR rely on negative samples, while BYOL eliminates them entirely.
- **Architectural Complexity**: MoCo uses a dynamic dictionary, while SimCLR simplifies the architecture, and BYOL introduces an online-target network mechanism.
- **Feature Separation**: SimCLRv2 and MoCov2 improve feature separation with projection heads and larger encoders, whereas BYOL relies on stable target networks.
- **Training Efficiency**: MoCo's dynamic dictionary and momentum updates provide efficient contrastive learning, while SimCLR's requirement for large batch sizes can slow down training.

4 Methodology

Building on the insights derived from the background analysis of both SSML and SSL models, this methodology first focuses on showcasing the SSML framework. By starting with SSML, we aim to thoroughly explore its computational processes, utilizing various DNNs to demonstrate the applicability and effectiveness of this framework. This approach provides a clear example of the unified computational process within SSML, illustrating how it enhances model performance and interpretability.

In subsequent sections, we shift our focus to SSL, applying similar principles to generalize the methodology. By doing so, we demonstrate the adaptability and scalability of the unified computation framework in different learning paradigms, thereby strengthening the overall contribution of the research to model-agnostic learning.

4.1 Analogy

In the context of XAI, our methodology is positioned within the post-hoc explanation paradigm. Specifically, it adopts a model-agnostic approach, allowing it to provide interpretability across different machine learning models without being tied to the specifics of any particular model architecture. By focusing on explanation through visualization, this research introduces the Unified Model-Agnostic Computation (UMAC) framework as a novel tool to interpret and enhance machine learning models. UMAC falls under the broader category of model-agnostic post-hoc explanations, which explain models by generating visual representations of their internal processes and outputs, as illustrated in Figure 1.

Through UMAC, we aim to provide clear and interpretable insights into the workings of both SSML and SSL models, enhancing transparency and explainability within these complex learning systems.

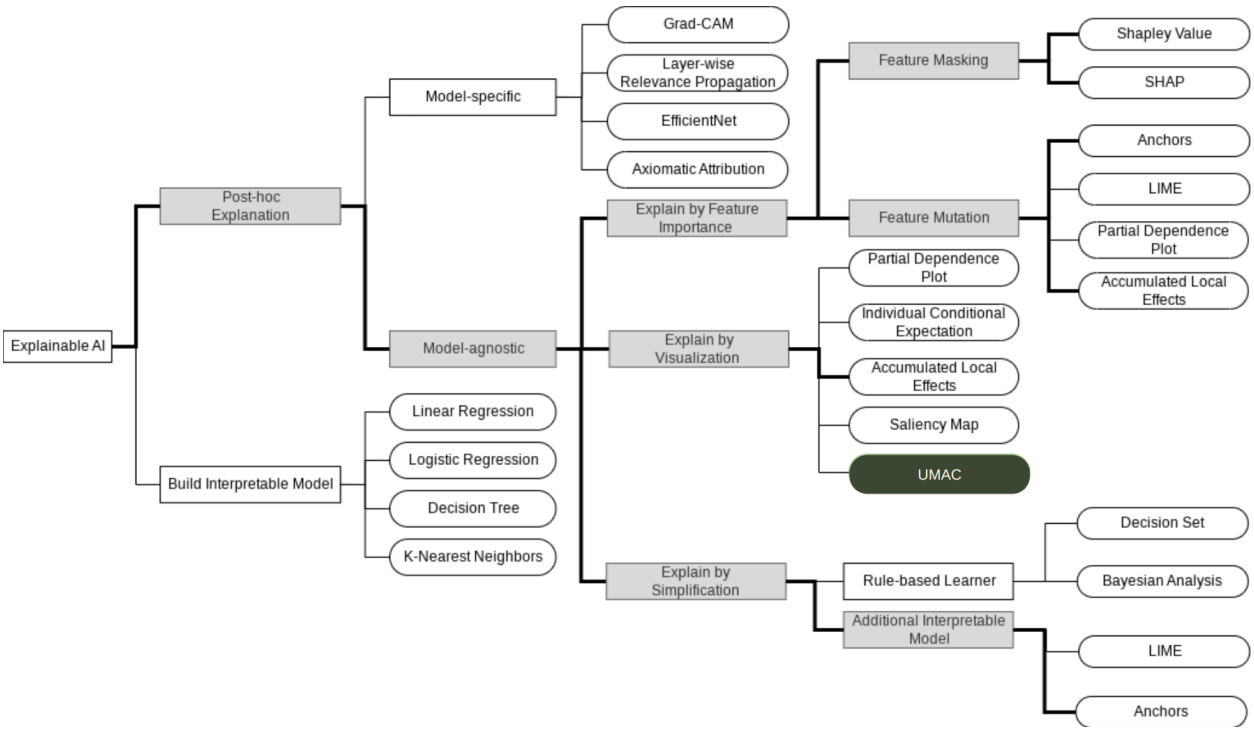

Figure 1: Positioning of UMAC within the XAI landscape

When reviewing the literature, particularly in the fields of SSML and SSL, the specific methods and techniques employed are frequently highlighted. However, a consistent and in-depth exploration of the intricate relationship between preprocessing and network classifier design is often

lacking. Given the importance of these components, this research seeks to bridge this gap.

To provide clarity, our study focuses on two key components:

- **Data Preprocessing Techniques**: Building on the foundational works by [Chen et al., 2020c, Grill et al., 2020], we explore the variety and depth of preprocessing techniques that have evolved over the years. These techniques play a critical role in influencing model performance, particularly in SSML and SSL scenarios. Our aim is to collate, compare, and analyze these methods to identify best practices and potential areas for improvement.

- **Network Classifier Design**: Diverse architectural decisions, as emphasized by [He et al., 2020], demonstrate the varied approaches researchers have taken. These range from simpler architectures to more complex, multi-layered designs, with significant implications for model accuracy, efficiency, and interpretability. We examine these choices to identify patterns, optimizations, and strategies for improving performance.

By juxtaposing these components against established literature, our goal is to provide a holistic understanding of their interplay. We aim to illuminate the symbiotic relationship between data preparation and classifier design, and how optimizing their interaction can enhance the performance and success of SSML and SSL models.

4.1.1 Preprocessing

Data preprocessing is a critical step in model training and serves as the foundational layer for any machine learning task [Witten et al., 2016]. Its components include the following:

- Data augmentation: Techniques like cropping, rotation, and flipping introduce variability, ensuring that the network is exposed to diverse patterns for better generalization [Shorten and Khoshgoftaar, 2019, Perez and Wang, 2017].

- Contrastive samples generation: In the context of contrastive learning methods, this discusses the mechanics of crafting ”positive” and ”negative” sample pairs to guide the network in discerning data relationships [Chen et al., 2020b].

- Normalization and scaling: This transforms data to a standard scale, ensuring that no particular feature disproportionately influences the model’s learning [Jain et al., 1996].

- Comparison across methods: Evaluating the preprocessing strategies in different self-supervised methods to distinguish nuances and similarities in their approaches [Zbontar et al., 2021].

The algorithm presented in Algorithm 2 serves as a powerful tool for enhancing SSL by augmenting input data. It systematically generates a diverse set of augmented images from an original input, thereby enriching the training dataset and improving the model's ability to recognize complex patterns without the need for explicit annotations.

Algorithm 1 Image Data Augmentation

Require: Input image X, number of augmentations n
Ensure: Augmented image sets $S_1, S_2, \ldots, S_n$
Initialize empty sets $S_1, S_2, \ldots, S_n$
for $i = 1$ to n **do**
  // Randomly select augmentation parameters
  $B_i, N_i, C_i, O_i \sim$ randomParameters()
  for $k = 1$ to k_{color} **do**
    // Randomly select subcolor transformation parameters
    $c_{ik} \sim$ randomSubcolorTransformation()
  end for
  for $k = 1$ to k_{spatial} **do**
    // Randomly select subspatial transformation parameters
    $o_{ik} \sim$ randomSubspatialTransformation()
  end for
  // Create and apply augmentation function
  $A_i \leftarrow$ createAugmentationFunction$(B_i, N_i, C_i, O_i,$
    $c_{i1}, c_{i2}, \ldots, c_{ik}, o_{i1}, o_{i2}, \ldots, o_{ik})$
  $X_i \leftarrow A_i(X)$
  S_i.add(X_i)
end for
return $S_1, S_2, \ldots, S_n$

The process begins with an initial image denoted as X and a specified number n that determines the volume of augmented iterations to be produced. The primary objective is to create n distinct sets of augmented images, each with its unique characteristics.

To achieve this, the algorithm employs a sequential procedure, iterating from 1 to n. During each iteration, a set of augmentation parameters is randomly selected. These parameters include the following:

- Blur intensity (B_i) and noise level (N_i): These parameters introduce subtle visual distortions,

which can enhance the model's robustness against slight image alterations.

- Color adjustments (C_i): Color adjustments modify the chromatic attributes of the image, ensuring that the model does not exhibit bias toward specific color palettes. Additionally, subcolor transformation parameters c_{ik} are selected randomly for finer adjustments, further diversifying the color variations within the augmented data [Zintgraf et al., 2017].

- Spatial transformations (O_i): Spatial transformations manipulate the spatial configuration of the image. This step is crucial for training the model to adapt to various object orientations and positions. Subspatial transformation parameters, o_{ik}, are also randomly selected, introducing diverse spatial alterations.

The randomization of these parameters is facilitated through the $R()$ function, ensuring a broad and comprehensive set of augmented images. The heart of this algorithm lies in the creation and application of the augmentation function A_i. This function incorporates the selected parameters, including B_i, N_i, C_i, and O_i, as well as the subcolor transformation parameters $c_{i1}, c_{i2}, \ldots, c_{ik}$ and subspatial transformation parameters $o_{i1}, o_{i2}, \ldots, o_{ik}$. By applying this function to the original image X, a transformed image, X_i, is obtained and stored within its corresponding set S_i. Upon completing all iterations, the algorithm yields a comprehensive suite of augmented images, represented as $S_1, S_2, \ldots, S_n$.

In summary, this algorithm is instrumental in generating a multitude of uniquely altered iterations of an original image. Each image undergoes a series of randomized modifications, encompassing blur, noise, color, and spatial adjustments, as well as subcolor and subspatial transformations. This process is essential for enhancing the robustness and efficacy of self-supervised learning models, as it provides a diverse and enriched dataset for training, thereby allowing models to better generalize and perform effectively in real-world scenarios.

4.1.2 Network Classifier

Following the schematic overview provided in Figure 2, we proceed to unpack the functionalities and significance of each component outlined earlier. In the forthcoming paragraphs, we delve into the intricacies of these elements, shedding light on their pivotal roles within the network classifier's

architecture. The network classifier is pivotal in the architecture of a machine learning model. It encompasses the following components:

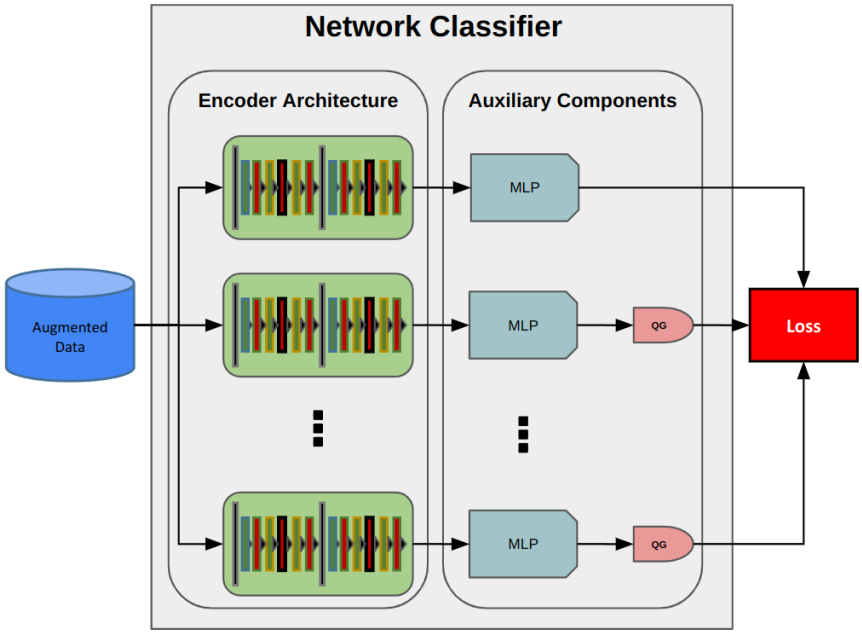

Figure 2: Network classifier's computational process

- **Encoder architecture:** The encoder is a fundamental component of the network classifier. Its primary function is to transform the input data into a representation that can be utilized effectively by the subsequent layers, as shown in Figure 2. The encoder's design, which is pivotal for the efficacy of SSL, defines how well the model can infer patterns from the input data. For instance, different input types such as images, text, or audio may necessitate unique encoder architectures. In the realm of SSL, encoders typically yield a dense representation, which is then channeled into a projection head to be refined further.

  For our studies, which focus predominantly on image classification, convolutional neural networks (CNNs) constitute the core of the encoder [587, 2016, He et al., 2015b]. The total parameter count P in a convolutional layer is expressed as

$$P = (F_W \times F_H \times D_{\text{in}} + 1) \times D_{\text{out}}$$

  where F_W and F_H denote the filter's width and height, D_{in} represents the depth of the input

volume, and D_{out} is the number of filters. The "+1" accounts for the bias term associated with each filter [Goodfellow et al., 2016a].

One crucial aspect while crafting the encoder for image classification tasks is determining the depth (D) and width (W) of the network. The encoder's total parameter count, P_{total}, can be approximated by the summation of parameters across all layers. A judicious equilibrium between D and W ensures computational efficiency combined with the capability to discern detailed image patterns, thus bolstering the self-supervised learning framework.

- **Auxiliary components:** Supplementary to the primary encoder are the auxiliary components, as illustrated in Figure 2. These elements bolster the encoder's capacity to deduce patterns from the input data. Within the self-supervised learning context, such components fine-tune the encoder's representation prior to its progression to the projection head. Notable among these are the following:

  - Multi-layer perceptron (MLP): This feedforward neural network comprises multiple node layers, each completely interconnected with the subsequent one. MLPs frequently serve as projection heads in self-supervised learning, refining the representations derived from encoders. The parameter count in an MLP can oscillate between iterations.
  - Queuing of representations: This data structure retains representations produced by the encoder. In self-supervised learning, it typically stores image representations, which are then employed as negative samples in the contrastive loss function. As the model undergoes training, the queue is continuously updated with new representations, while simultaneously discarding the oldest to maintain a consistent size.

- **Number of encoders in network classifier:** In the design of a network classifier, the choice of the number of encoders is pivotal for model performance and interpretability [Pascanu et al., 2013]. Typically, the architecture leans towards using two encoders, seldom more. The underlying reason for this is grounded in the mathematics of data transformation and the "multiple vector problem".

  Consider that each encoder E_i transforms the input data X into a representation space R_i. Mathematically, this can be represented as

$$R_i = E_i(X; \theta_i)$$

where θ_i denotes the parameters of the i-th encoder. The challenge arises when these representation spaces, especially for $i > 2$, start becoming either too overlapping (redundant) or too disjointed (losing coherence). The ideal scenario is for the representation spaces to be distinct yet complementary.

Furthermore, when encoders increase beyond two, the combined transformation function can be visualized as [Doe and Smith, 2023]

$$R = f(E_1(X; \theta_1), E_2(X; \theta_2), \ldots, E_n(X; \theta_n))$$

This intricate composition intensifies the "multiple vector problem" [Doe and Smith, 2023]. In essence, data representations are pushed in various directions in the high-dimensional space, leading to the potential challenge of ensuring that the final representation R remains meaningful and informative for downstream tasks.

The decision to use two encoders provides a balance. It allows the model to diversify the representation space, capturing different facets of the data, but without the complications of handling multiple potentially conflicting directions. The gradients during backpropagation, represented by ∇R, remain more stable, mitigating issues associated with deep architectures like vanishing or exploding gradients [Pascanu et al., 2013].

4.2 Development of Unified Model-Agnostic Computation for SOTA Models

Developing a unified model-agnostic computation (UMAC) system requires a structured methodology that integrates diverse computational models, algorithms, and frameworks efficiently and scalably. The steps of this process are illustrated in Figure 3. The aim is to create a computation system that is versatile enough to handle various data types and computational environments, while also being capable of incorporating the latest advancements in methodologies. This process involves selecting, analyzing, and designing computation processes for SOTA algorithms from a variety of machine learning paradigms.

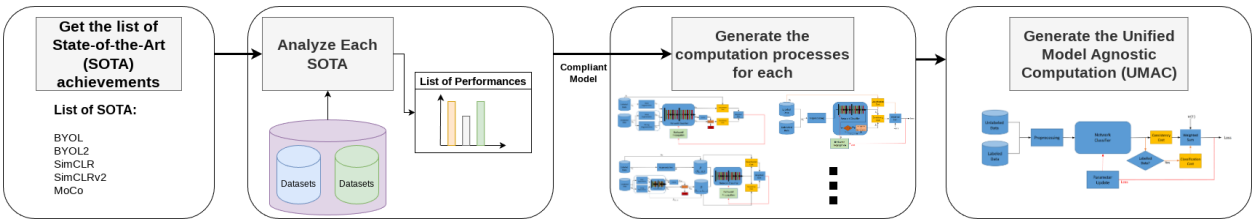

Figure 3: Process to define the unified model-agnostic computation

- **Replication of existing results**: We first replicate the results reported by the original authors of each SOTA algorithm using their datasets and evaluation metrics. This allows us to establish a baseline for comparison and ensure that our implementation is accurate.

  - **Outcome**: The replication step serves as the foundation for building trust in the UMAC framework. By reproducing the results of existing SOTA algorithms, we ensure that our implementation aligns with recognized benchmarks before making further adjustments.

- **Tweaking algorithmic features**: After replication, we begin modifying the behavior of these algorithms by tweaking various parameters, such as learning rates, optimization techniques, and data preprocessing methods. This enables us to explore the impact of different configurations on performance and efficiency.

  - **Outcome**: Tweaking allows us to evaluate the flexibility of each SOTA algorithm and identify which parameter adjustments yield the best performance improvements. This is key for ensuring that UMAC can accommodate various architectures and settings without losing effectiveness.

- **Interchanging components between SOTA solutions**: In cases where different SOTA solutions share similar features (e.g., preprocessing steps, model architectures, or loss functions), we experiment by interchanging these components between solutions. This allows us to test the compatibility and adaptability of the algorithms and identify potential synergies between them.

  - **Outcome**: Interchanging components helps validate the modular nature of UMAC, confirming that different parts of algorithms can be seamlessly swapped while maintaining or even improving performance. This reinforces UMAC's plug-and-play capability and its adaptability to a variety of computational models.

- **Testing for compliance with our criteria**: During this experimental phase, we continuously test each modified algorithm against the criteria established in the previous section (performance, scalability, generalizability, innovation, and community recognition). This ensures that the SOTA solutions remain aligned with the objectives of our research and meet the required standards.

  - **Outcome**: This step ensures that only the most suitable SOTA algorithms, those that meet our rigorous standards, are integrated into UMAC. It helps in maintaining the robustness, efficiency, and generalizability of the framework, ensuring its alignment with the research objectives.

We will showcase each of these steps for the SOTA solutions in the subsequent subsections.

4.2.1 Identify the SOTA for each specific area

This step involves gathering a comprehensive list of SOTA algorithms from a variety of machine learning paradigms, including SSML, SSL, supervised, and unsupervised learning. We also investigate application-specific domains where these SOTA solutions have demonstrated success, ensuring that we cover a broad range of areas such as medical imaging, natural language processing, and autonomous systems.

In this stage, we focus on the following criteria for identifying SOTA solutions:

- **Performance metrics**: We consider algorithms that have achieved significant improvements in key performance indicators such as accuracy, precision, recall, and F1 score within their respective fields.

- **Scalability**: The SOTA algorithms should demonstrate the ability to scale efficiently with increasing data size and complexity.

- **Generalizability**: Algorithms must show strong generalizability across different datasets and applications, proving their robustness and adaptability.

- **Innovative techniques**: Preference is given to solutions that introduce novel techniques or improvements in computational methods, such as model architectures, training processes, or data handling techniques.

- **Community recognition**: We focus on algorithms that have been peer-reviewed and accepted by leading academic and industrial conferences or journals, indicating recognition by the research community.

This initial phase allows us to not only gather information about the leading algorithms but also understand the ”know-how” behind their success. This information is reflected in the Background section and forms the foundation for the solutions considered in our research.

4.2.2 Analyze each SOTA solution

This step involves assessing each identified SOTA solution's performance and tracking how it evolves and improves over time. Our goal is to evaluate the enhancements that each algorithm introduces, comparing performance metrics such as accuracy, precision, recall, and computational efficiency. However, this phase goes beyond mere evaluation.

In this stage, we also begin experimental setups where we replicate and tweak each SOTA solution to understand its behavior under different conditions. This involves the following tasks:

- **Replication of existing results**: We first replicate the results reported by the original authors of each SOTA algorithm using their datasets and evaluation metrics. This allows us to establish a baseline for comparison and ensure that our implementation is accurate.

- **Tweaking algorithmic features**: After replication, we begin modifying the behavior of these algorithms by tweaking various parameters, such as learning rates, optimization techniques, and data preprocessing methods. This enables us to explore the impact of different configurations on performance and efficiency.

- **Interchanging components between SOTA solutions**: In cases where different SOTA solutions share similar features (e.g., preprocessing steps, model architectures, or loss functions), we experiment by interchanging these components between solutions. This allows us to test the compatibility and adaptability of the algorithms and identify potential synergies between them.

- **Testing for compliance with our criteria**: During this experimental phase, we continuously test each modified algorithm against the criteria established in the previous section (perfor-

mance, scalability, generalizability, innovation, and community recognition). This ensures that the SOTA solutions remain aligned with the objectives of our research and meet the required standards.

Through this iterative process of testing and tweaking, we not only assess each SOTA solution's inherent strengths and weaknesses but also explore opportunities for enhancement by combining features from different algorithms. This phase helps us build a deeper understanding of how these solutions can be refined and adapted for integration into the unified computational framework.

4.2.3 Design computational processes for each solution

This step involves structuring the computational processes for each SOTA solution to understand how they achieve their performance gains. The goal is to model the internal workings of these algorithms while ensuring that the integrity of the original models remains intact—meaning no changes or modifications are made to the core structure of the models themselves.

To achieve this, we design the computation processes with the following considerations:

- **Maintain model integrity**: We ensure that the core architecture and components of the original SOTA models remain unchanged to preserve their intended functionality and performance characteristics.
- **Develop computational graphs**: For each solution, we create a detailed computational graph that visualizes the entire workflow of the model. This includes key components such as data input, preprocessing, layers within the model, loss functions, and optimization techniques.
- **Illustrate model functionality**: The computational graph serves to depict how different elements of the model interact, providing insights into the flow of data and how it contributes to performance gains. This visualization enhances understanding without altering the model's internal structure.

The computational graph allows us to represent the interaction between different parts of the model, offering insights into the model's behavior without altering its underlying structure. This

approach ensures that we preserve the fidelity of each SOTA solution while providing a clear, interpretable representation of its computational processes.

For example, consider a SOTA semi-supervised learning (SSML) algorithm like the Mean-Teacher model. In this case, the computational graph would illustrate how the teacher model and the student model interact over multiple iterations. Specifically, it would highlight how the weights of the teacher model are updated through exponential moving averages (EMA) of the student model's weights, without modifying the architecture itself. In Figure 6 in Section 4.3.3, the inputs from labeled and unlabeled datasets are shown entering both the student and teacher models. The outputs differ: the student model calculates the classification loss from the labeled data, while the consistency cost is derived from the difference between the student and teacher predictions on the unlabeled data. This visual breakdown helps to dissect the model's learning process, showing how the two losses (classification and consistency) contribute to the overall training, without altering the underlying architecture.

By using such a computational graph, we can maintain the integrity of the SOTA solution while allowing machine learning engineers to better understand where performance bottlenecks may occur and where potential improvements can be made.

4.2.4 Develop the UMAC system

Finally, this step synthesizes the insights gathered from analyzing the SOTA solutions into a comprehensive UMAC system. The UMAC system provides a global explanation of the enhancements and improvements made across various algorithms over time, offering a high-level understanding of the entire model's behavior, rather than focusing on individual predictions. As a global XAI method, UMAC captures the overarching patterns, structures, and interactions between different model components, which is essential for understanding how a model functions holistically and identifying key factors influencing its performance [Adadi and Berrada, 2018].

As such, the development of the UMAC system involves several key requirements:

- **Generalization of computational processes**: The UMAC system is a generalization of the computational processes designed for each solution. It integrates insights from individual models to create a broader, unified computational framework.

- **Multiple UMACs per SOTA**: In some cases, there may be more than one UMAC for a given solution. This ensures that different aspects of complex models can be captured without oversimplifying their functionality. Each UMAC is tailored to represent specific computational features of the model.

- **Joint UMAC representation**: The system is designed to include joint UMACs where appropriate, meaning we create a unified computation that reflects the interactions between different processes when it fits the model's behavior. This prevents overgeneralization while ensuring comprehensive coverage of the model's operations. See section 4.4 for more details.

- **Balancing complexity and simplicity**: We aim to strike a balance between overgeneralization and oversimplification. The UMAC system should be detailed enough to capture the critical aspects of the model's performance, yet simplified enough to remain interpretable and applicable across different solutions. Maintaining this balance is crucial to preserving the integrity of the model.

- **Preserving model integrity**: Similar to the individual computational processes, the UMAC system is developed with the goal of maintaining the original model's integrity. The system is designed to mirror the core functionality and relationships within the model, ensuring that the UMAC accurately reflects the model's true behavior.

In this way, the UMAC system serves as a comprehensive and adaptable model-agnostic framework that captures both the individuality and commonalities across various SOTA solutions, ensuring transparency and interpretability without sacrificing the integrity or complexity of the models.

In the following subsections, we will demonstrate the steps for designing computational processes for each solution, using SSML as an example. Subsequently, we will showcase the development of the UMAC system for these models.

4.3 Generation of SSML Model Specific Computation Processes

4.3.1 Temporal Ensembling

The temporal ensembling structure [Laine and Aila, 2016] maintains the same configuration as the generic as the main SSML, featuring a network classifier with a single DNN. The primary distinction between Temporal Ensembling and the generic form with multiple DNNs lies in the fact that, here, network augmentations are assessed only once per epoch, and network consistency relies on previous evaluations for the unsupervised loss component.

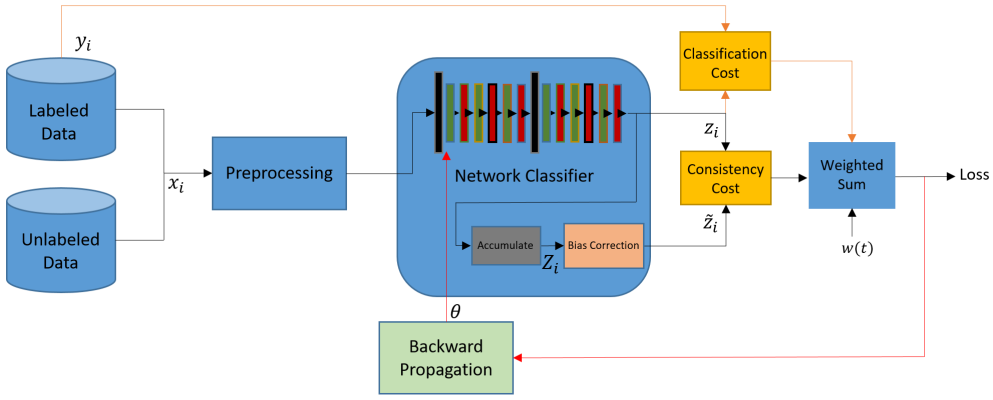

Figure 4: Representation of the Temporal Ensembling Model Structure

In the Temporal Ensembling model, the data x_i undergoes the same preprocessing steps to achieve optimal performance. After this phase, the preprocessed data is fed into a Neural Classifier network, which determines the classification prediction z_i. In the subsequent iteration, z_i is aggregated, aiding in the computation of both Consistency and Classification Costs in the current iteration. This mirrors the process followed in Generic SSML; however, the computation of Consistency Cost is exclusive to the unlabeled data. In Figure 4, the critical path for various steps, in the case of unlabeled data, is illustrated as a black path. For labeled data, both Classification and Consistency Costs are calculated, and the forward propagation is depicted by the black and orange paths. Moreover, the total loss computation is articulated in Equation 13, mirroring that of the

generic SSML as show here:

$$\begin{aligned} \text{Loss} &= \text{Classification Cost} + \text{Consistency Cost} \\ &= \text{ Cross Entropy } + w(t) * \text{ Squared Difference} \\ &= -\frac{1}{|B|} \sum_{i \in (B \cap L)} y_i * \log z_i + w(t) * \frac{1}{C|B|} \sum_{i \in B} \|z_i - \tilde{z}_i\|^2 \end{aligned} \tag{1}$$

Where,

z_i specifies the $f_\theta\left(g\left(x_{i \in B}\right)\right)$ and θ specifies the function trainning parameter.

$|B|$ represents the size Batch

L includes the labelled input indices

y_i the label for input x_i

$w(t)$ specifies the time-dependent weighting function (it is discussed later)

C specifies the number of different classes $y_i \in \{1 \ldots C\}$ (cat, tree, dogs, car etc..)

The Classification Cost computation aligns precisely with that of the generic SSML, detailed in Equation 13, and is defined as Cross Entropy between the prediction z_i and the label y_i.

Regarding Consistency Cost, despite being governed by the same Equation 13, there's a significant variation in the target vector $\tilde{z}_i$. Here, $\tilde{z}_i$ isn't derived from another Neural Network's assessment under varying dropout conditions. Instead, it is an accumulation of ensemble predictions, preceded by a bias correction step. The Neural Network computes z_i, which is then amalgamated into the ensemble outputs Z_i following the update $Z_i \leftarrow \alpha Z_i + (1 - \alpha) z_i$ [Laine and Aila, 2016], with α representing the momentum term that dictates the depth of historical training data influencing the ensemble. During the initial iteration ($i = 0$), z_{i-1} is set to zero, indicating that the initial values for $\tilde{z}$ and Z are also zero. Consequently, it's imperative to correct the startup bias in Z by dividing it by the factor $(1 - \alpha^t)$ to obtain the training target $\tilde{z}_i$. In the context of Backward Propagation, this loss is applied solely to the current Neural Network.

Temporal Ensembling offers several advantages over other SSML algorithms, as outlined below:

- Since the network undergoes evaluation once for each input, the training process is expedited.

- With a training target characterized by reduced noise compared to the Π-model, Temporal Ensembling can potentially realize superior accuracy performance.

However, Temporal Ensembling also presents certain limitations, detailed as follows:

- It necessitates the storage of auxiliary data and the introduction of a new hyperparameter, α, thereby demanding additional memory resources.
- Similar to the Π-model, the Temporal Ensembling model lacks dynamic learning capabilities, necessitating system retraining for effective functionality.

4.3.2 Π-model

This model, a variant of SSML, employs a self-ensembling mechanism during training [Laine and Aila, 2016], ensuring network output consistency across two instances of the same input under different dropout conditions.

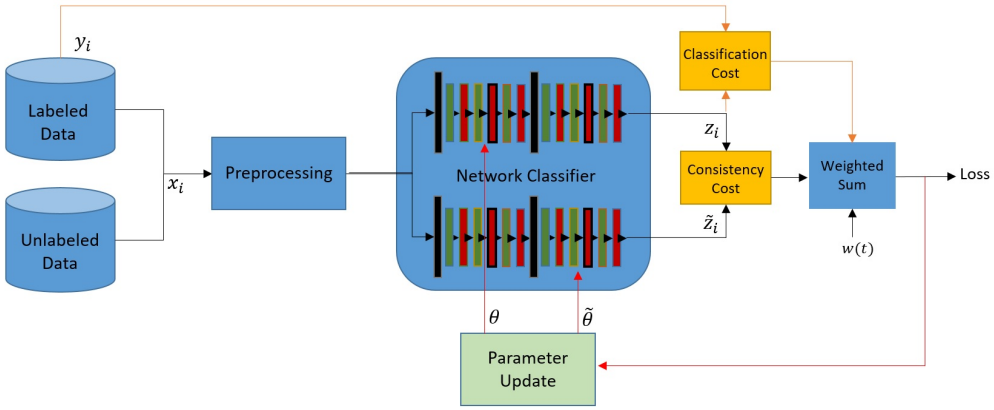

Figure 5: Π-model Structure Representation

In Figure 5, both labeled and unlabeled data x_i are subjected to identical preprocessing, without discrepancy. Upon entry into the Network Classifier, the data is replicated across two analogous neural networks. These networks are differentiated by the following factors:

- Dropout regularization, infusing an element of randomness into these twin architectures.
- The types of augmentations employed, which expand the dataset and render identical labeled and unlabeled data as distinct inputs.

These two Neural Networks yield the classifications z_i and $\tilde{z}_i$, which are instrumental in computing the total loss. When labeled data is input into the Network Classifier, both Consistency and Classification Costs are derived. As shown in Figure 5, this scenario necessitates following the black and orange paths for forward propagation to ascertain the total loss.

Conversely, for unlabeled data, only the Consistency Cost is derived, meaning that the black path in Figure 5 is pursued during forward propagation. Ultimately, the losses across the entire dataset are aggregated via a weighted sum. A pivotal aspect here is the employment of the ramp-up function $w(t)$, which assigns weights to the Consistency Costs, compelling the semi-supervised system to prioritize Classification Costs in the initial stages. Throughout the loss computation, a backward propagation must be initiated to update the parameters aimed at loss reduction. In Figure 5, backpropagation is signified by the red path, indicating that both Neural Network parameters, θ and θ', are updated, thus revising the entire neural network flow.

Evaluating the performance of the Π-model, we observe that it offers better accuracy than preceding SSML algorithms. However, this model has the following limitations:

- The network output is computed twice for the same input data, resulting in additional computational costs.
- The model is vulnerable to errors when fed incorrectly labeled data, negatively impacting its accuracy.
- This model cannot learn dynamically as it relies on batch-learning, contrasting with online learning neural networks.

4.3.3 Mean Teacher

To address the shortcomings of two SSMLs given in the previous subsections, the Mean Teacher method calculates the averages of the model weights rather than predicting the outcome. This model is based on an average of consecutive student models; hence, it is referred to as the Mean Teacher method [Laine and Aila, 2016]. In this case, the raw data x_i is first preprocessed, and then injected into the Network Classifier in the same manner as the previous SSML models. However, in this case, the Network Classifier is not the same as previous models. Here, the Network Classifier consists of two identical DNNs with Dropout Conditions referred to as the Student and

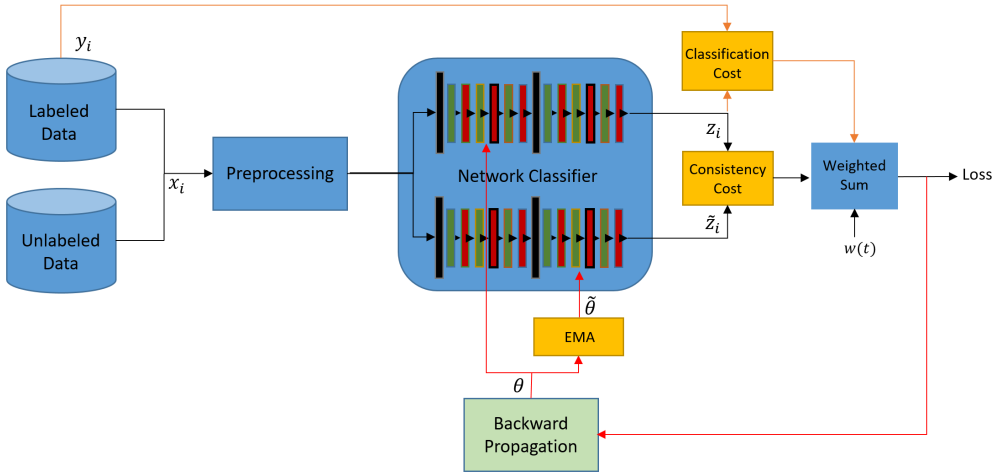

Figure 6: Mean Teacher Model Structure Representation

Teacher Models. Primarily, the Mean-Teacher model is exactly the same as the $\Pi-$Model during the computation of forward propagation for both unlabelled and labelled data. However, the Mean Teacher method diverges from the $\Pi-$Model during the backward propagation phase and in the strategy it employs to update weights.

The backward propagation is simply computed for the the parameters θ of the Student Model. For the Teacher Model, the parameter update for $\tilde{\theta}$ happens using the parameters of the Student Model θ by performing an EMA. This can be observed in Equation 2 [Laine and Aila, 2016]:

$$\tilde{\theta}_t = \alpha\tilde{\theta}_{t-1} + (1-\alpha)\theta_t \tag{2}$$

Where,

$\tilde{\theta}_t$ specifies the parent weights

$\tilde{\theta}_{t-1}$ signifies the parent weights of the previous run

θ_t specifies the child weights.

α specifies the smoothing parameter.

It has been observed that averaging of the model weights over various training steps is more likely to provide a more accurate model compared to utilizing the final weights promptly. In case of Mean Teacher model, the information can be aggregated after each step rather than every epoch. However, previous research studies have overlooked a significant metric, which is the difference in the program's run-time. It is observed that the Mean Teacher model can be trained in much

lesser time than the Π-model, whereas it achieves much improved accuracy compared to Temporal Ensembling. However, there are some drawbacks of the Mean Teacher model:

- Both the supervised networks perform a forward propagation using different parameters. However, Student Model is the only one with backward propagation to compute the gradients.
- The Mean Teacher model utilizes much higher memory resources compared to the other two models since the Dropout conditions (η and η') must be kept and preserved during each epoch.

4.3.4 MixMatch

Compared to the previous methods discussed, MixMatch has a different approach. In every batch, they are Augmenting the raw labelled data x_b and keeping the labels y_b generating the dataset $\hat{X}$. For the Unlabelled data, the raw input x_b is augmented K times where K is a hyper-parameter. Then those K augmented entries are passed through the Network Classifier to predict all their labels $\tilde{z}_{b2}, \tilde{z}_{b2}$ till $\tilde{z}_{bk}$ and after that their average is taken $\tilde{z}_{bavg}$ as a prediction for all the K entries. $\tilde{z}_{bavg}$ is sharpened using temperature sharpening to reduce the entropy of the label distribution. This can be done as follows:

$$\text{Sharpen}(p, T)_i := \frac{p_i^{\frac{1}{T}}}{\sum_{j=1}^{L} p_j^{\frac{1}{T}}} \tag{3}$$

Where,

p is some input categorical distribution
T is a hyperparameter that needs to be tuned
L is the number of labelled classes.

As such, the output of the sharpening will produce the unlabelled data with the pseudo-labels $\bar{z}_b$ denoted as $\hat{U}$. Concatenating and shuffling both $\hat{X}$ and $\hat{U}$, we get the W. MixUp is done on the $\hat{X}$ and the first $|X|$ entries of W to get X' where $|X|$ is the size of the labelled data in the batch. As well MixUp is applied on the unlabelled data in the batch $\hat{U}$ with the rest of the entries W to get U'. To understand how MixUp is working we need to understand how (x', p') is computed using

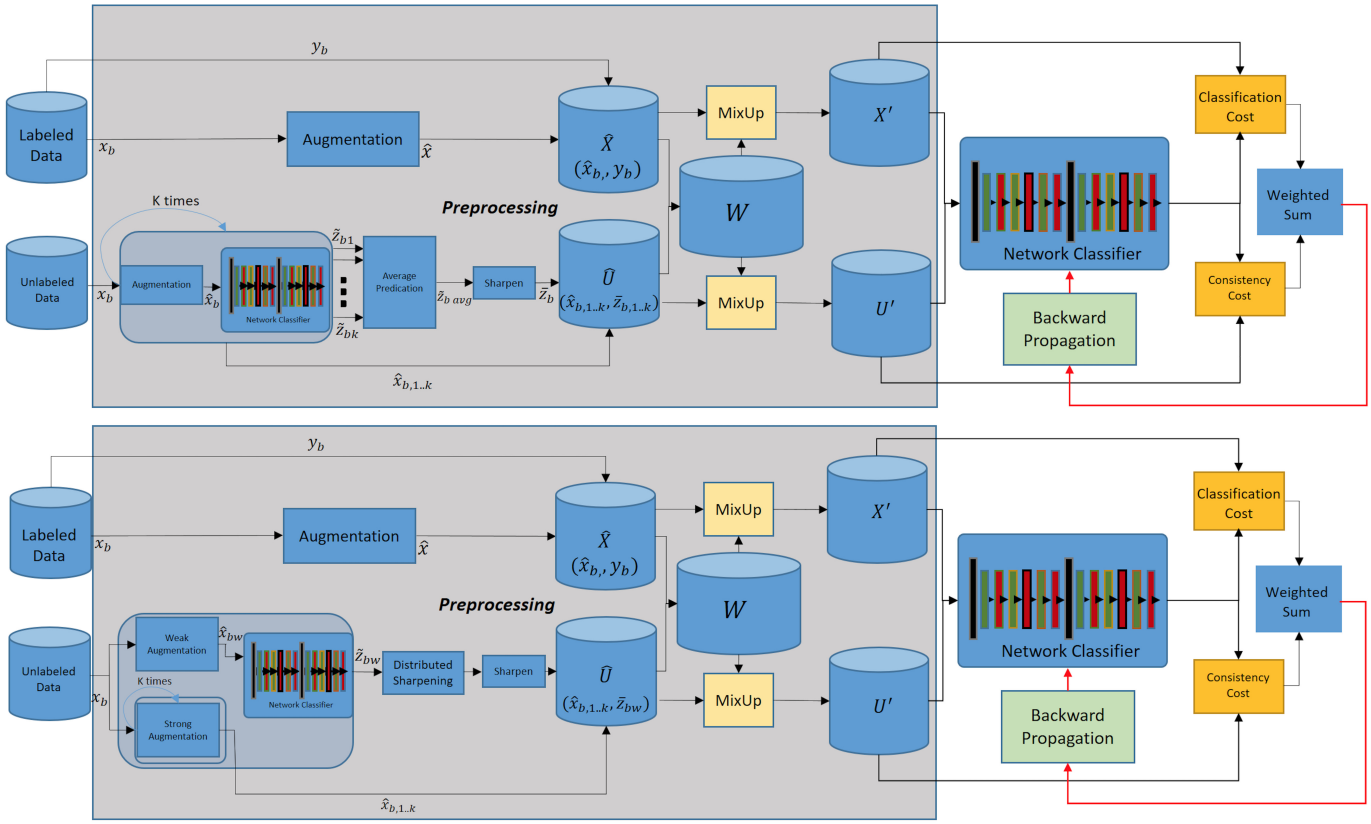

Figure 7: MixMatch (Top) and ReMixMatch (Bottom) Model Structure Representation

two examples with their corresponding labels probabilities $(x_1, p_1), (x_2, p_2)$ as shown below:

$$\lambda \sim \text{Beta}(\alpha, \alpha) \tag{4}$$

$$\lambda' = \max(\lambda, 1 - \lambda) \tag{5}$$

$$x' = \lambda x_1 + (1 + \lambda')x_2 \tag{6}$$

$$p' = \lambda p_1 + (1 + \lambda')p_2 \tag{7}$$

Where α is hyperparameter to tune. Due to $\lambda \geq 0.5$ and the max function, the first term x_1 and it's label p_1 gets more importance than the second point x_2 and it's label p_2. This makes MixUp model prediction for X' and U' correspond to labelled and unlabelled output guesses respectively. As we know the exact output of labelled data, the classification cost is basically a cross-entropy. On the other hand, for the consistency cost since we are guessing the labels for the unlabelled data, L_2 loss is used. These losses can be shown in equation below:

$$L_X = \frac{1}{|X'|} \sum_{x,p \in X'} H(p, p_{model}(y|x;\theta)) \tag{8}$$

$$L_U = \frac{1}{L|U'|} \sum_{x,p \in U'} ||\bar{z}_b - p_{model}(y|u;\theta))||_2^2 \tag{9}$$

$$L = L_X + \lambda_U L_U \tag{10}$$

4.3.5 ReMixMatch

Following the same principle as MixMatch, ReMixMatch adopts the new method but introduces several modifications. One significant alteration is the augmentation anchoring method. As depicted in Figure 7, the labeled input x_b undergoes a robust augmentation process using a variant of AutoAugment known as CTAugment, which concurrently learns an augmentation policy during model training. This strongly augmented labeled data $\hat{x}_b$, together with the label y_b, constitutes the dataset $\hat{X}$. For the unlabeled input x_b, the model employs multiple strong augmentations K, resulting in $\hat{x}_{b,1..k}$. Using the same unlabeled raw input x_b, a weakly augmented version is generated

by applying only a crop or a flip, yielding $\hat{X}_{bw}$. This $\hat{X}_{bw}$ is then processed through the Network Classifier to obtain the prediction $\tilde{z}_{bw}$.

Another notable modification is the implementation of distribution alignment on $\tilde{z}_{bw}$. This adjustment involves maintaining a running average of the Network Classifier predictions for the unlabeled data, denoted as $\tilde{p}(y)$, and estimating the marginal class distribution $p(y)$ for the labeled examples during training. The model then applies a ratio of $p(y)/\tilde{p}(y)$ to the Network Classifier's prediction of the label $z = p_{model}(y|u;\theta)$, followed by normalization using $\text{Normalize}(x)_i = x_i / \sum_j x_j$. Subsequently, Temperature Sharpening, as previously discussed in the context of MixMatch, is applied to produce the pseudo-labels $\bar{z}_{bw}$.

Utilizing both $\hat{x}_{b,1..k}$ and $\bar{z}_{bw}$, the model generates the Unlabeled dataset $\hat{U}$. It is evident that the multiple strongly augmented versions of the unlabeled data are based on the weakly augmented variant. Beyond this stage, the procedure aligns precisely with that of MixMatch, adhering more generally to the framework.

4.4 Unified Model Agnostic Computation for SSML

As with many other ML algorithms, SSML algorithms maintain a similar structure, albeit with minor differences. Referring to Figure 8, we observe the network classifier-focused structure of previous SSML models, such as the Π model, Mean Teacher, and Temporal Ensembling. The preprocessing of labeled and unlabeled data is conducted, which are then injected into the network classifier without splitting, simultaneously. Throughout the preprocessing step, data quality is enhanced by suppressing undesirable distortions or enhancing certain image features essential for further processing. For instance, some of these operations could include filtering, geometric transformations, pixel brightness adjustments, or a combination thereof.

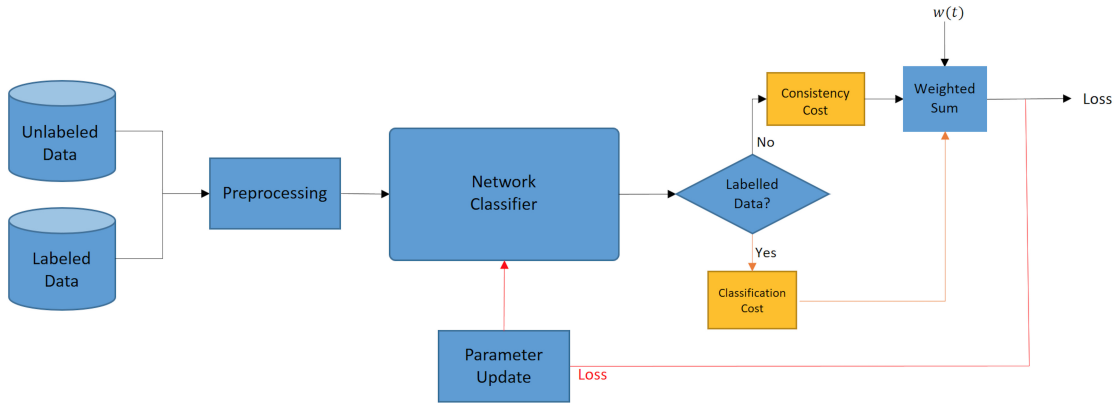

Figure 8: Network classifier-focused semi-supervised computation process (CF-SSCP).

Analyzing these SSML algorithms, the network classifier block comprises single or multiple instances of essentially the same DNN to execute the optimal data classification. Consequently, this unique network classifier design facilitates transformative parameter updates. This feature differentiates various SSML models. A network classifier can include the following types:

- One DNN: Here, a single DNN is employed in the classifier, conducting both backward and forward propagations.
- Two DNNs: This configuration utilizes two nearly identical DNNs, differing slightly in their dropout conditions and parameter initialization. Forward propagation occurs in both DNNs; however, backward propagation might not occur in one of these DNNs in some designs.
- Three or more DNNs: Here, three or more DNNs are employed, potentially with varying initialization conditions. Additionally, dropout conditions may or may not be integrated into these DNNs.

Depending on the total number of DNNs, the network classifier's output integrates one or more classifications for computing the loss. Based on the data type input, there are two distinct cases for loss computation:

- Unlabeled data: For unlabeled data, the consistency cost among two or more DNN instances is calculated, multiplied by the number of classes and the inverse of the batch size.
- Labeled data: In this scenario, the consistency cost among two or more DNN instances and the classification cost between one or more models are computed.

It is noteworthy that the classification cost is calculated only for labeled data. For unlabeled data, the classification cost is not computed and is set to zero.

Upon assessing the most recent state-of-the-art MixMatch and ReMixMatch, we noticed a deviation in the structure of the previous semi-supervised computation process. These alterations are depicted in Figure 9.

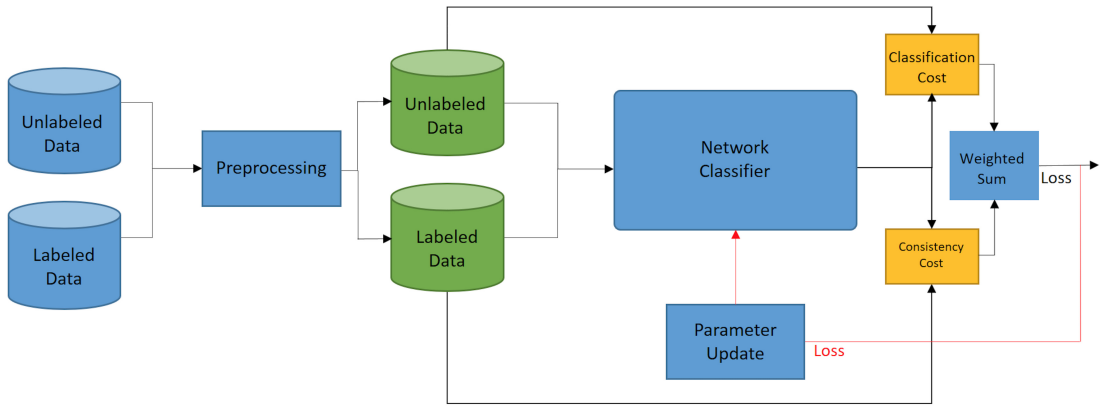

Figure 9: Preprocessing-focused semi-supervised computation process (PF-SSCP).

As illustrated, preprocessing is applied to both unlabeled and labeled datasets, subsequently outputting postprocessed unlabeled and labeled datasets. The preprocessing step generates pseudo-labels for the unlabeled datasets, effectively rendering both datasets labeled [587, 2016]. Regarding the network classifier, in most designs, it is simplified to one DNN; however, there is potential for further exploration. The network classifier's predictions are utilized for calculating the classification cost and the consistency cost. Examining each SSCP, the primary distinctions between PF-SSCP and CF-SSCP are as follows:

- The design's emphasis is placed on preprocessing steps rather than the network classifier.
- The classification cost is consistently calculated due to the pseudo-labels on the unlabeled datasets.

Furthermore, the consistency cost (also known as the unsupervised loss) can be computed using the classification outputs of one or more DNN instances, denoted as z_i and $\tilde{z}_i$. Moreover, in instances where only one DNN is employed, $\tilde{z}_i$ represents the evaluations of the previous network,

not a second network's evaluation. Generally, the consistency cost is formulated as follows [Laine and Aila, 2016, Murphy, 2013]:

$$\begin{aligned} \text{Consistency Cost} &= w(t) * \text{ Squared Difference} \\ &= w(t) * \frac{1}{C|B|} \sum_{i \in B} \|z_i - \tilde{z}_i\|^2 \end{aligned} \tag{11}$$

where

z_i represents $f_\theta\left(g\left(x_{i \in B}\right)\right)$ and θ denotes the function's training parameter.
$\tilde{z}_i$ denotes $f_{\theta'}\left(g\left(x_{i\ inB}\right)\right)$, and θ' represents the function's training parameter.
$|B|$ is the batch size.
$w(t)$ signifies the time-dependent weighting function (discussed subsequently).
C represents the number of classes, with $y_i \in \{1 \ldots C\}$ (e.g., cat, tree, dog, car, etc., for CIFAR-10).

When data are labeled, z_i is used in conjunction with the label y_i to compute the classification cost (or the supervised loss). The supervised loss, or classification cost, is essentially the log of the product of classification and the corresponding label. For mini-batches of size B, this is the inverse of the mini-batch size multiplied by the sum of the losses' negative values, as shown in Equation (12) [Laine and Aila, 2016]:

$$\text{Classification Loss } = -\frac{1}{|B|} \sum \log z_i[y_i] \tag{12}$$

Combining both losses provides the total loss used for updating the network's parameters. In light of these computations and the observed variations in SSCP designs, the following research question emerges:

$$\begin{aligned} \text{Loss } &= \text{Classification Cost} + \text{Consistency Cost} \\ &= \text{ Cross Entropy } + w(t) * \text{ Squared Difference} \\ &= -\frac{1}{|B|} \sum_{i \in (B \cap L)} y_i * \log z_i + w(t) * \frac{1}{C|B|} \sum_{i \in B} \|z_i - \tilde{z}_i\|^2 \end{aligned} \tag{13}$$

where

z_i specifies the $f_\theta\left(g\left(x_{i\in B}\right)\right)$ and θ specifies the function training parameter.

$|B|$ represents the batch size.

L includes the labeled input indices.

y_i is the label for input x_i.

$w(t)$ specifies the time-dependent weighting function (discussed later).

C specifies the number of different classes $y_i \in \{1 \ldots C\}$ (cat, tree, dogs, car, etc..)

5 Semi-Supervised Computation Processes (SSCPs) for SSML

As we developed the UMAC framework for SSML, it is crucial to assess its performance and understand how it can be applied in various settings. In this section, we examine the performance of UMAC within the context of SSML, focusing on how it facilitates the interpretation of complex models, such as those involving Convolutional Neural Networks (CNNs). By applying UMAC, we aim to enhance transparency, optimize model performance, and bridge the gap between explainability and prediction accuracy in SSML algorithms.

Next, we will explore the specific methodology employed to evaluate these aspects through a series of empirical experiments.

5.1 Empirical Evaluation and Experimental Design

Before diving into the experiments and assigning values to specific hyperparameters, it is important to discuss the methodology used to address the research questions. The adoption of various techniques for applying XAI methods [Adadi and Berrada, 2018, Arrieta et al., 2019] to analyze trained models has steered XAI towards post-hoc analysis. It is essential to remember that the complexity of a machine learning model is often inversely related to its interpretability [Chen et al., 2018]. Generally speaking, the more complicated and unconstrained a model is, the more difficult it is to interpret or explain clearly [Lipton, 2018]. Thus, integrating CNNs within SSML algorithms results in non-monotonic and non-linear response outputs, contributing to the creation of models that are among the least interpretable. In this context, our methodology is model-specific, concentrating on global measures that will enable us to comprehend fully the inputs and their complex modeled correlations with the output predictions.

5.1.1 Datasets

In this section, we present the dataset utilized for training SSML classifiers. The CIFAR-10 dataset was selected, comprising 60,000 distinct 32 $\times$ 32 color images evenly distributed across ten different classes, with each class containing 6000 images. From this dataset, we allocated 10,000 images for the testing process and the remaining 50,000 images for the training phase.

In terms of the dataset, CIFAR-10 was selected for its simplicity and its common use in benchmarking the chosen state-of-the-art SSML classifiers. We reduced the labeled data in various ratios primarily to address **RQ1.2** while also evaluating the comparison of PF-SSCP and CF-SSCP on aspects such as training time, training loss, and learning accuracy, thereby contributing to the resolution of **RQ1.1**. Given our objective of testing the SSML algorithms with varying quantities of labeled data, we intentionally withheld some of the labels during the training of these SSML algorithms, as detailed in Table 18.

Table 4: CIFAR-10 Training Datasets.

Labeled Data	**Unlabeled Data**	**Labeled Data per Class**	**Unlabeled Data per Class**
1000	49,000	100	4900
4000	46,000	400	4600
50,000	0	5000	0

As depicted in Table 18, both labeled and unlabeled data are proportionately distributed across the ten classes, indicating an impartial approach with no class preference. The substantial gap between the 4000 and the 50,000 labeled datasets serves as an ideal measure for assessing the efficacy of the unsupervised components relative to the supervised elements within various SSML algorithms. In the case of the 1000 labeled datasets, the SSML algorithms undergo rigorous testing, primarily relying on the unsupervised components of the network, thereby approaching the realm of unsupervised machine learning algorithms [He et al., 2020].

5.1.2 Preprocessing

The preprocessing step primarily hinges on the specific SSML approach and the dataset in use. For the purposes of this paper, preprocessing is tailored to either the PF-SSCP or the CF-SSCP method-

ologies. In CF-SSCP, we execute zero component analysis (ZCA) [587, 2016], a step adopted by both the Mean Teacher and Π models in their processing of the CIFAR-10 datasets, which contributes to enhanced accuracy and diminished loss. Conversely, preprocessing in the context of PF-SSCP, as demonstrated in strategies such as MixMatch and ReMixMatch, adopts a more intricate approach previously elaborated upon in Sections 4.3.4 and 4.3.5. This method capitalizes on the network classifier's capability to generate pseudo-labels, negating dependence on randomized algorithms. Additionally, a crucial element in both preprocessing-focused and classifier-focused SSCP is data augmentation, a strategy that significantly betters results by enriching the training dataset. We executed various data augmentation techniques, including image flips, zooms, shifts, and cropping.

5.1.3 SSML and DNN Combination

The choice of SSML model dictates the nature of integration to be employed, a necessity given the distinct behaviors exhibited by different architectural choices [Szegedy et al., 2015]. This combination evaluates five critical aspects:

- Performance, gauged through accuracy and loss metrics.
- The duration of the training process.
- Constraints imposed by hardware.
- The initial selection of hyperparameters.
- The appropriateness of the loss function.

Both PF-SSCP and CF-SSCP are guided by these metrics, though the degree of influence each one holds varies. This variance is attributable to specific design features outlined in the preceding section for each SSML type and their respective behaviors. In the context of CF-SSCP, most DNNs utilize dropout regularization, a technique instrumental in preventing network overfitting. Conversely, PF-SSCP does not mandate overfitting prevention measures within the DNNs.

5.1.4 Ramp-Up and Ramp-Down Functions

In CF-SSCP, a ramp-up period was initiated with 40,000 training steps at the onset of the training process. During this phase, both the learning rate and the consistency cost parameters were progressively increased from zero to their peak values using a sigmoid-shaped function. Conversely, for PF-SSCP, a linear ramp-up was employed, escalating from an initial value of 100 to the maximum over the initial 16,000 training steps.

5.1.5 SSML Performance Measurements

This segment details the approach adopted to assess the classification efficacy of the SSML networks, with a specific focus on the variety of network classifiers involved in the evaluation. As indicated earlier, our experimental analysis was confined to the CIFAR-10 dataset. Per the discussion in Section 4.4, any SSML algorithm can be abstracted into two distinct types of SSML architectures, as illustrated in Figures 8 and 9, by employing diverse network classifiers. In this study, we restricted our consideration to contemporary CNN models serving as the network classifiers in SSML networks. For the training and evaluation phases of the proposed systems, we utilized a GeForce GTX 2080 Ti GPU. This specific GPU was chosen for its proficiency in handling an array of SSML algorithm and network classifier combinations [Hadjis et al., 2016].

It is pertinent to mention that in CF-SSCP, for a streamlined analysis, we amalgamated the preprocessing stage with the feature selection process, as both these procedures precondition the datasets prior to their introduction to the network classifiers. In contrast, PF-SSCP placed a greater emphasis on the preprocessing stages as well as the generation of pseudo-labels, owing to the inherent design of PF-SSCP, to more significantly affect the dataset rather than the network classifier.

The ensuing performance metrics were established as the foundation for the appraisal of the proposed models:

- Performance accuracy and training loss, evaluated against varying proportions of unlabeled to labeled datasets.
- The duration of training necessitated by each amalgamation of SSML with diverse network classifiers, inclusive of the time expended during the preprocessing and feature selection stages.

- Utilization of parameters and the complexity intrinsic to various SSML algorithms, with an exposition on the distinct parameters requisite in each scenario and their subsequent influence on complexity.

5.1.6 Experimental Design and Framework Specifications for SSML

In order to rigorously evaluate the performance of semi-supervised learning models, our experiments incorporate two principal computational frameworks: the CF-SSCP and the PF-SSCP frameworks, as outlined in Table 5. These frameworks provide the basis for comparing models by assessing the impact of classifier complexity and preprocessing sophistication. The table showcases the relevant experiments conducted, presenting a clear framework for analysis, while a comprehensive list of experimental hyperparameters and additional experimental details are described in our publication [Neghawi and Liu, 2023]. Additional experiment setup details can be found in the appendix A.2.2, which provides extended descriptions of the configurations and methodologies used throughout the experiments, further enriching the context for replication and comparison.

Table 5: Experiments for different semi-supervised models under CF-SSCP and PF-SSCP frameworks.

Framework	Model	Experiment and Architecture	Summarized Purpose
CF-SSCP	Temporal Ensembling (TE)	Exp 1: TE and Shake-Shake26	Ensemble predictions over time for stability and consistency
		Exp 2: TE and DenseNet-121	Utilizing ensemble learning with a focus on network depth
		Exp 3: TE and WRN-40-2	Applying ensembles to widen and deepen network architectures
		Exp 4: TE and WRN-28-10	Ensemble methods combined with a wider network model
	Π Model	Exp 5: Π model and Shake-Shake26	Ensuring consistent network predictions without skip connections
		Exp 6: Π model and DenseNet-121	Consistency of prediction with depth-oriented network architectures
		Exp 7: Π model and WRN-40-2	Deeper networks under consistency constraints
		Exp 8: Π model and WRN-28-10	Wider networks maintaining prediction consistency
	Mean Teacher (MT)	Exp 9: MT and Shake-Shake26	Teacher–student model consistency without skip connections
		Exp 10: MT and DenseNet-121	Depth and skip connections in a teacher–student setup
		Exp 11: MT and WRN-40-2	Deeper architecture in a mean teacher framework
		Exp 12: MT and WRN-28-10	Enhanced width in the teacher–student model's architecture
PF-SSCP	MixMatch (MM)	Exp 13: MM and Shake-Shake26	Augmentation and mixing strategies for semi-supervised learning
		Exp 14: MM and DenseNet-121	Deep architecture applied to advanced mix-and-match techniques
		Exp 15: MM and WRN-40-2	Widening and deepening networks with semi-supervised mix–matching
		Exp 16: MM and WRN-28-10	Wide network structures in advanced mix–match learning scenarios
	ReMixMatch (RM)	Exp 17: RM and Shake-Shake26	Refinement of mix–match techniques with a preprocessing focus
		Exp 18: RM and DenseNet-121	Application of preprocessing strategies in deep learning models
		Exp 19: RM and WRN-40-2	Preprocessing alignment in wider and deeper network structures
		Exp 20: RM and WRN-28-10	Extensive preprocessing in a widened network scenario

The selection of architectures for our experiments was deliberate to encompass a broad range of complexities and capacities pertinent to semi-supervised learning:

- **DenseNet-121 (Dense Convolutional Network—121 layers):** Known for its dense connectivity, DenseNet-121 optimizes parameter efficiency and facilitates feature propagation and

reuse. Its design is particularly beneficial for learning with limited labeled data, which is a common challenge in semi-supervised learning scenarios [Huang et al., 2016a].

- **Shake-Shake Regularization Model (Shake-Shake26):** The Shake-Shake regularization approach, exemplified by the Shake-Shake26 model, introduces stochasticity [Mania et al., 2016] into the training process. This method has been shown to enhance generalization on image classification tasks, presenting a unique advantage in semi-supervised learning frameworks [Gastaldi, 2017].

- **Wide Residual Networks (WRN-40-2):** The WRN-40-2 architecture augments the network's width, offering an optimal trade-off between depth and width. This expanded network capacity allows it to represent more complex functions and data relationships, benefiting from the additional unlabeled data in semi-supervised learning setups [Zagoruyko and Komodakis, 2017].

- **WRN-28-10:** The WRN-28-10 extends the width of traditional residual networks even further, targeting the rigorous demands of high-complexity classification tasks. The architecture is designed to capitalize on the unlabeled data that are more prevalent in semi-supervised learning contexts [Zagoruyko and Komodakis, 2017].

The adoption of these architectures in our experiments allows us to thoroughly assess the efficacy of semi-supervised learning strategies across varying levels of network complexity and depth.

5.2 Analysis and Discussion for SSML

In this subsection, we provide a comprehensive analysis of the frameworks, comparing the performance of CF-SSCP and PF-SSCP methods. This analysis is centered on the models' accuracy, loss and training time. Our evaluation examines how the differences in focus—classifier versus preprocessing—impact the performance of SSML frameworks when applied to varying quantities of labeled and unlabeled data.

5.2.1 CF-SSCP with Various Network Classifiers

Given CF-SSCP's substantial reliance on the network classifier, it becomes pertinent to investigate its performance dynamics across different network classifiers. Accordingly, our tests for CF-SSCP were conducted using both single (Mean Teacher and Temporal Ensembling) and double (Π model) network classifiers. The ensuing subsections detail our findings.

Results of Temporal Ensembling: The Temporal Ensembling method, utilizing a singular DNN within the network classifier, stands as the most straightforward among the three SSML algorithms. As evidenced in Figures 10 and 11, it is apparent that, under Temporal Ensembling, the efficacy of both WRN-40-2 and DenseNet-121 is overshadowed by the more robust Shake-Shake26 model.

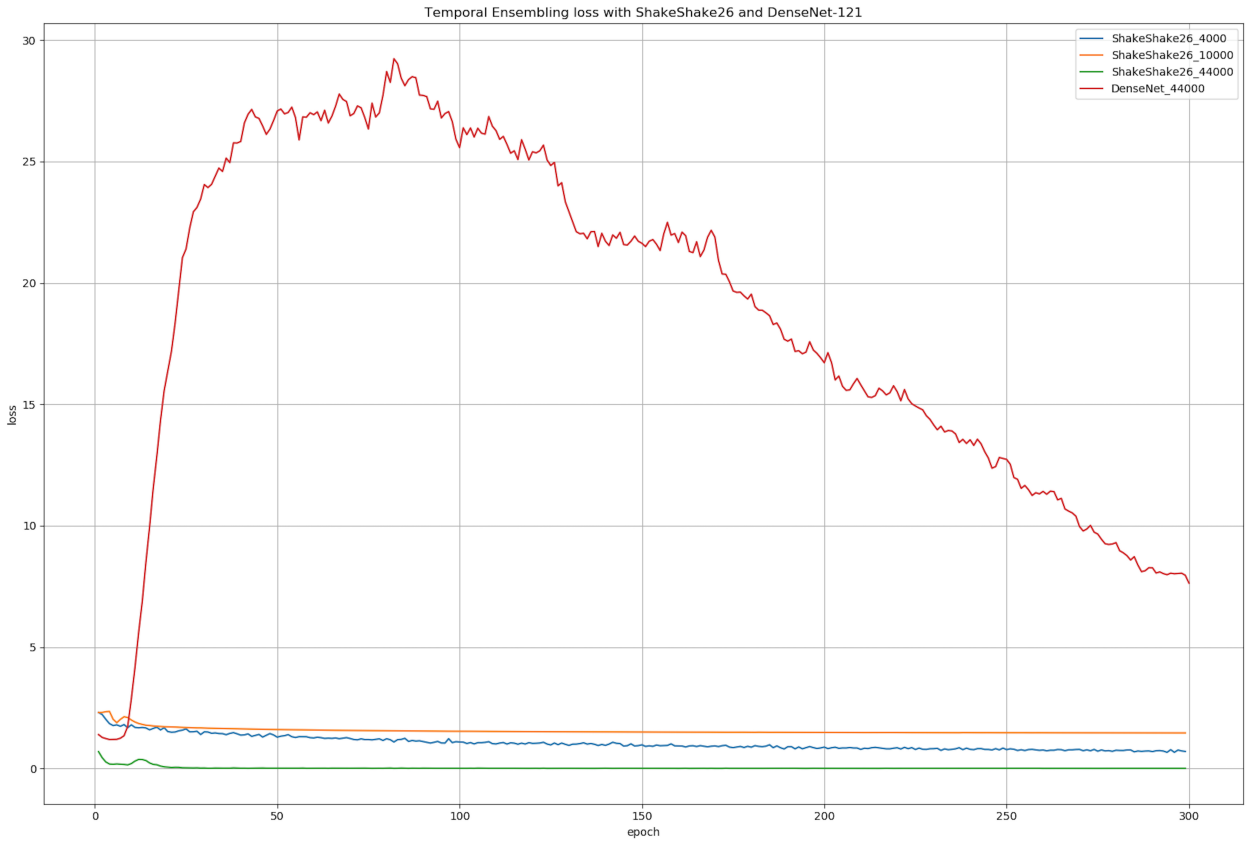

Figure 10: Loss comparison for Temporal Ensembling: DenseNet-121 vs. Shake-Shake26.

Figure 11: Loss comparison for Temporal Ensembling: WRN-40-2 vs. Shake-Shake26.

Inspection of the accuracy data in Figure 12 reveals that training accuracy marginally surpasses test accuracy, with the highest accuracy observed at 4,000 labels.

Figure 12: Training and testing accuracy for Temporal Ensembling with Shake-Shake26.

However, as Figure 13 demonstrates, the training loss for Shake-Shake26 within the Temporal Ensembling framework fails to reach optimal performance even with extensive hyperparameter tuning. This outcome likely stems from the model's simplicity, concentrating solely on the training loss of a single DNN. Thus, exploration into the alternative models, namely the Π model and Mean Teacher model, is warranted.

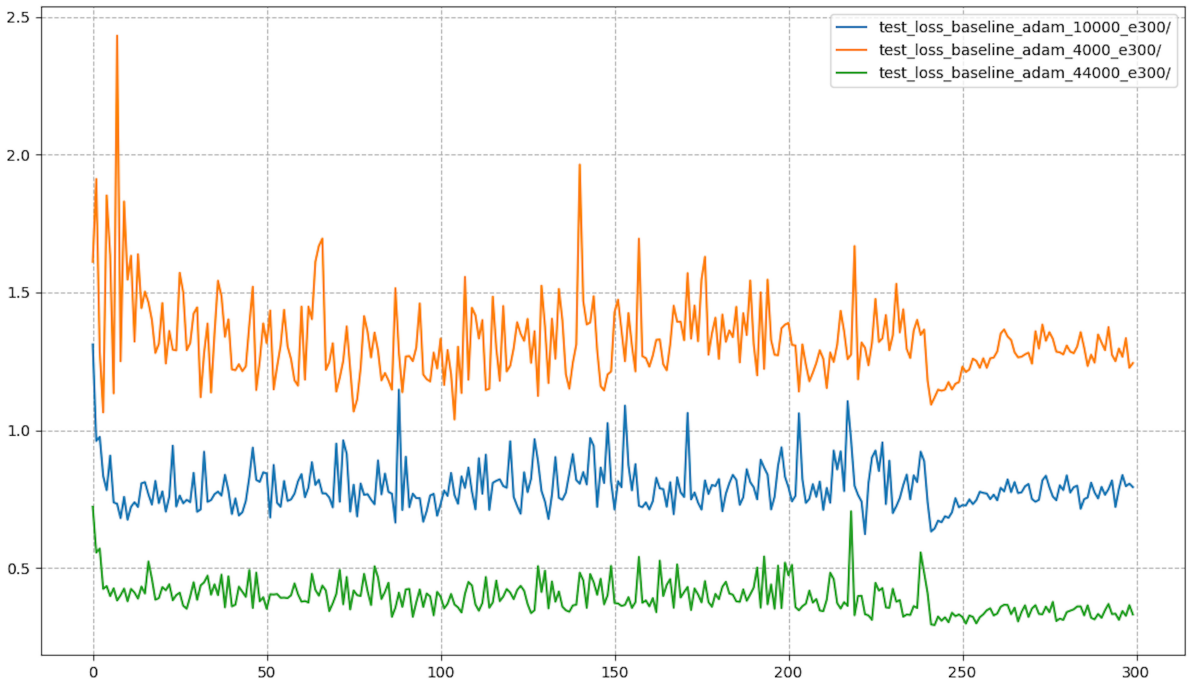

Figure 13: Training loss for Temporal Ensembling with Shake-Shake26.

Summary:

In all three experiments, it is evident that Temporal Ensembling experiences the highest loss compared to the other two models, the Πmodel and the Mean Teacher model. Analyzing the efficacy of various DNNs within the network classifier of Temporal Ensembling reveals that Shake-Shake26 incurs the least loss compared to the other two DNNs, namely, DenseNet and WideResNet. This suggests that a DNN with fewer skip connections can yield more accurate results and lower training loss. Additionally, employing two DNNs in a network classifier appears to facilitate better parameter updates during the computation of the consistency cost.

Results of Π Model and Mean Teacher: We conducted training and validation for the Π model and Mean Teacher model utilizing WideResNet as the primary network. This setup was designed to exhibit the performance metrics of each model by leveraging the CIFAR-10 dataset with varying percentages of labeled data.

- **WideResNet Core Network with Mean Teacher**: Figure 14 depicts the accuracy achieved by the Mean Teacher model using WideResNet as our core network. The graph indicates that

the highest validation accuracy was achieved using the Stochastic Gradient Descent (SGD) [Tieleman et al., 2012] algorithm with 44,000 labels, the maximum in these experiments. Notably, accuracy diminishes as the label count decreases. This trend was anticipated since accuracy correlates with the total label count, assuming unchanged hyperparameters.

However, the primary observation here concerns the discrepancy between the SGD and Adam optimizers. Clearly, the SGD optimizer [Goyal et al., 2017] surpassed Adam in terms of implementation accuracy.

Examining the loss in Figure 15, the Adam optimizer performs superiorly, with less test loss. Another notable aspect is both networks' initial struggle to smoothly reduce loss, resulting in a highly noisy transition of the loss function. This is attributed to the unsuitability of maintaining consistent hyperparameter values across different volumes of labeled data.

Another critical point pertains to initialization. Different initialization parameters are crucial, evident from the 10,000-label dataset starting significantly lower than the 44,000-label set, causing the latter higher initial loss and hindering performance.

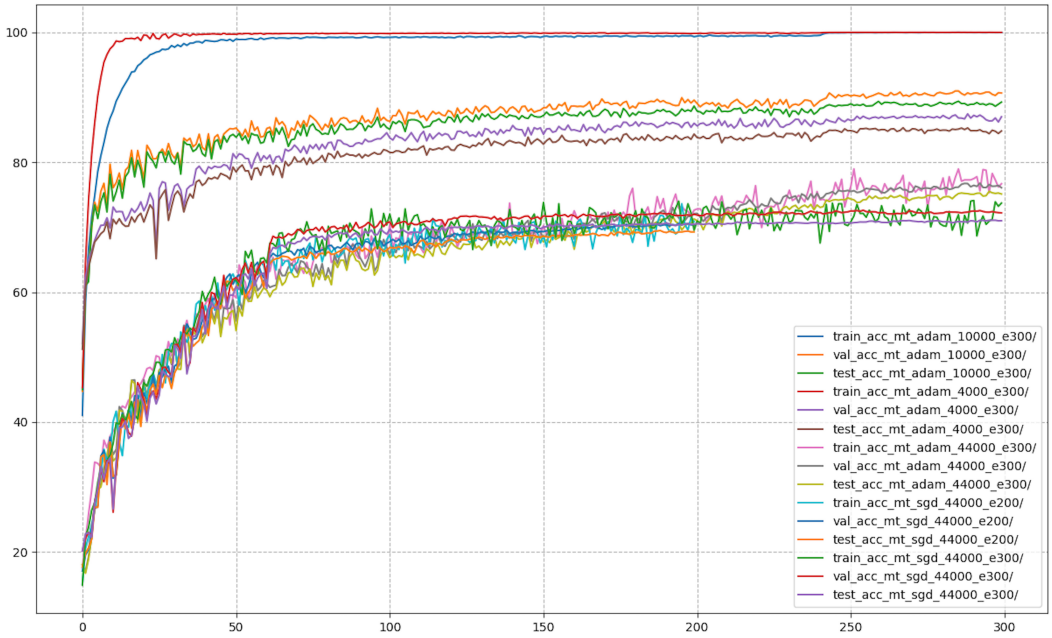

Figure 14: Mean Teacher accuracy with WideResNet across all label quantities.

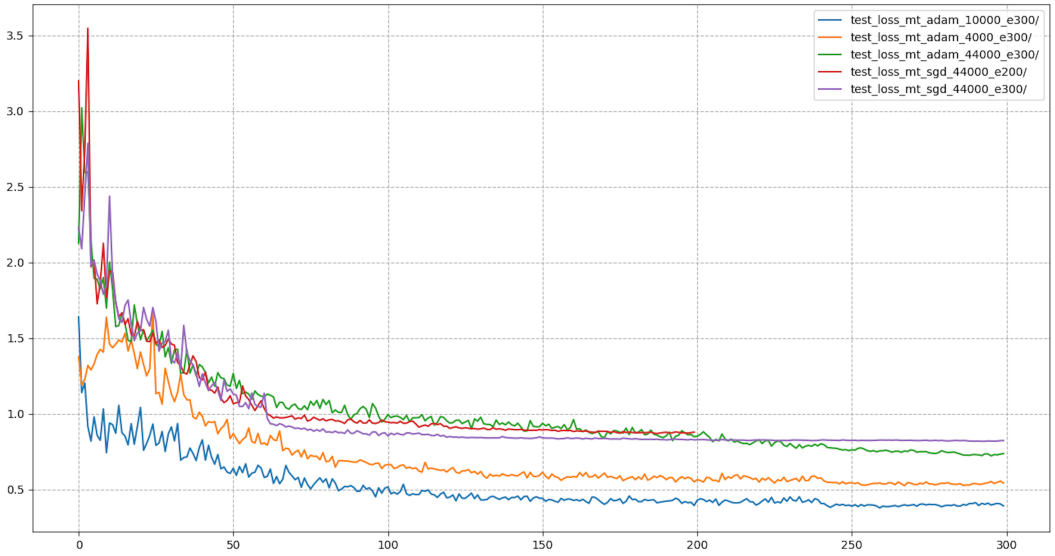

Figure 15: Mean Teacher loss with WideResNet across all label quantities.

- **WideResNet core network with Π model**: For the Adam optimizer, accuracy does not directly correlate with the number of labels, as distinctly seen in Figures 14 and 16.

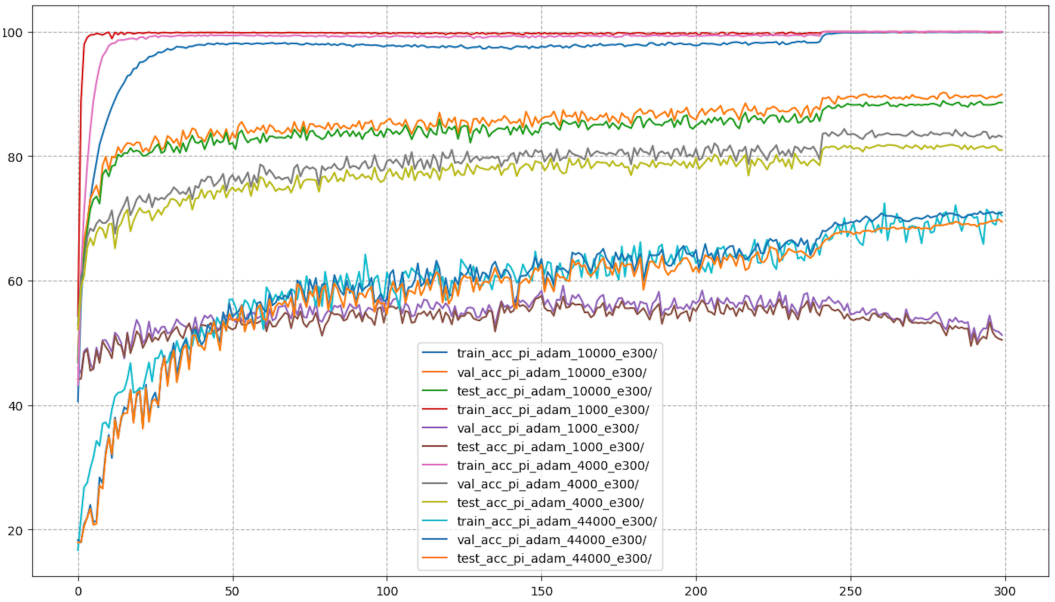

Figure 16: Π model accuracy with WideResNet across all label quantities.

Here, high training accuracy is noted with 1000, 4000, and 44,000 labels. Among these, the 1000-label dataset provides the highest accuracy. However, the 10,000-label dataset's training accuracy is 20% lower than the others, yet its testing accuracy, at 88.56%, surpasses

the rest. Additionally, the 1000-label set displays an improvement in accuracy up to 58% before a decline, suggesting overfitting, as evidenced in Figure 17.

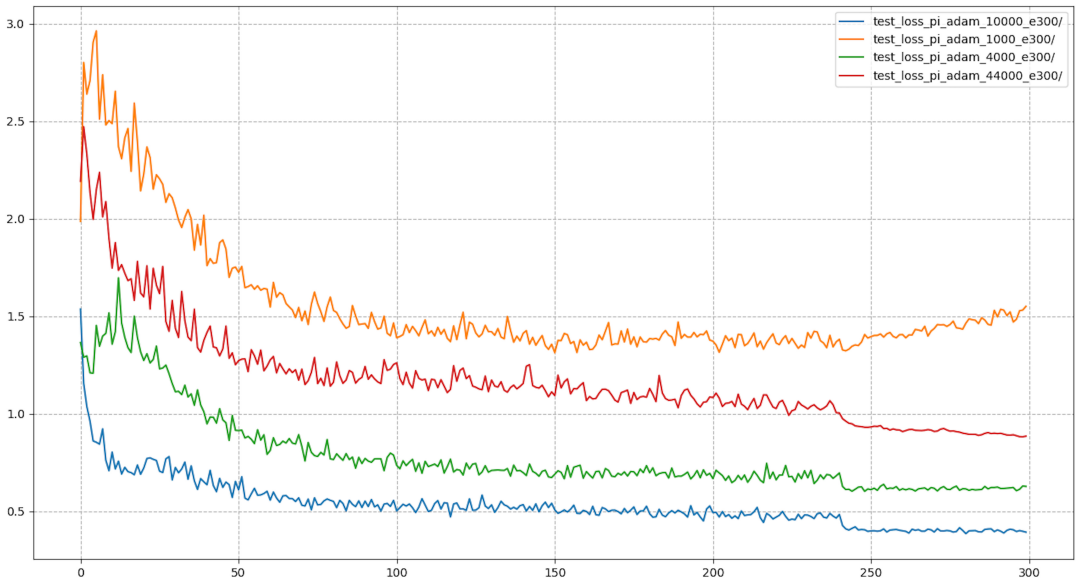

Figure 17: Π model loss with WideResNet across all label quantities.

Comparing this network with different label quantities, the loss is significantly lower at 10,000 labels, a result of hyperparameter tuning and network initialization. Even with 44,000 labels, the network struggles to reduce loss due to an unfavorable start.

Moreover, the network's commencement varies across datasets, except for the 10,000-label set, where loss increases from 0 to 30 epochs. Significant fluctuations are particularly noticeable for the 1000-label data. These issues could be addressed through hyperparameter tuning using strategies such as Random Search or Grid Search.

- **Comparison of Mean Teacher with Π model using WideResNet as the core network**: Figure 18 compares the testing and validation accuracy of both networks, demonstrating the Mean Teacher model's superiority over the Π model with 4000-label datasets.

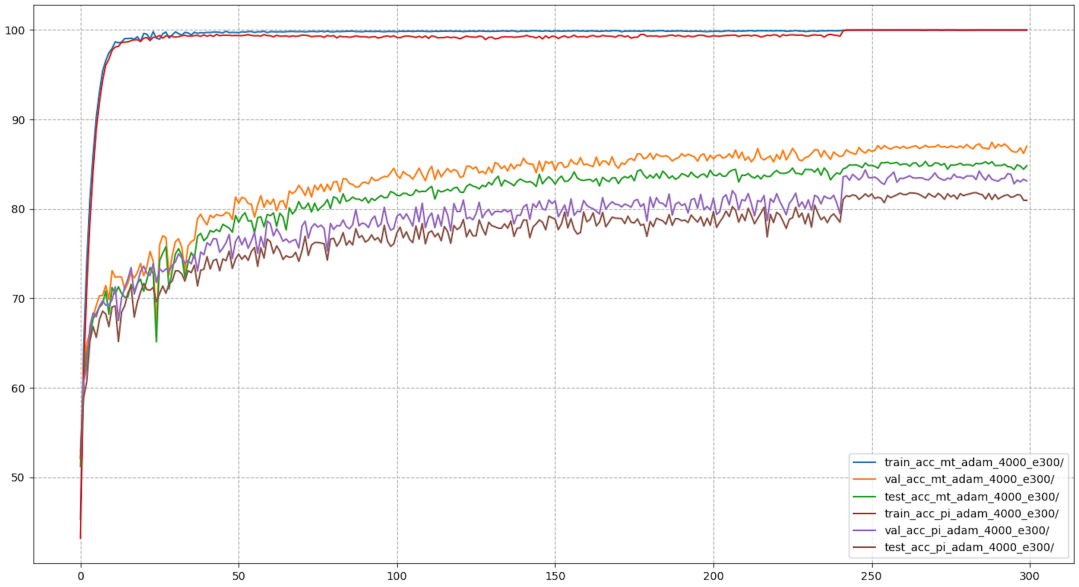

Figure 18: Mean Teacher vs. Π model in WideResNet accuracy at 4000 labels.

Furthermore, as shown in Figure 19, the Mean Teacher model excels in terms of loss, even with suboptimal hyperparameter tuning, highlighting the efficacy of the network design with the same core network and identical hyperparameters.

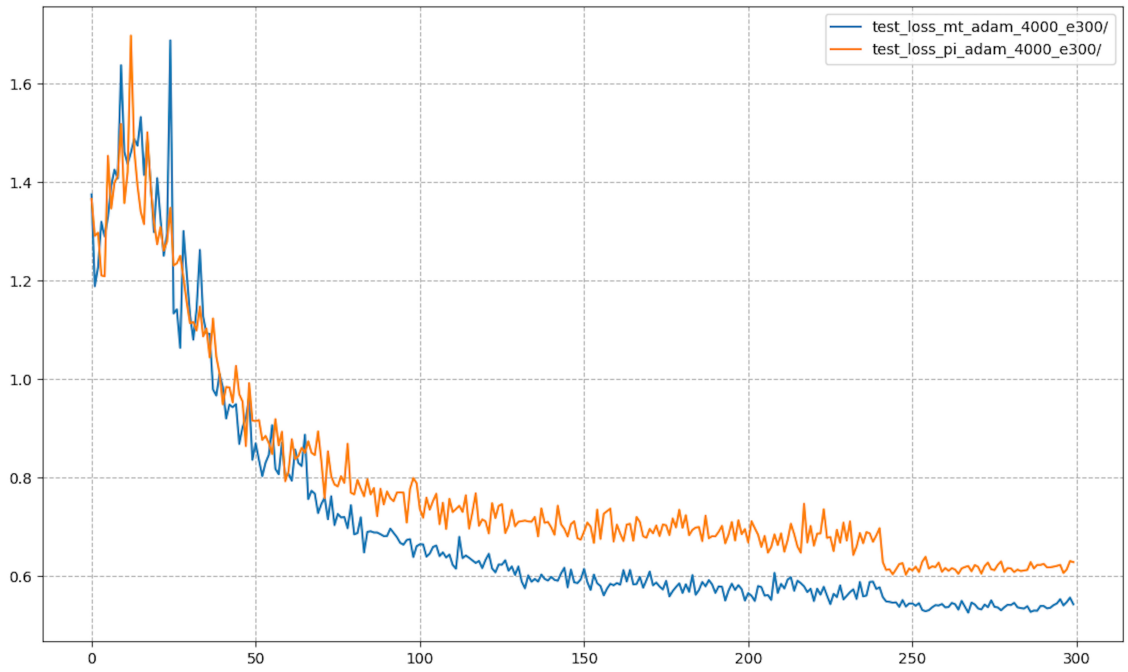

Figure 19: Mean Teacher vs. Π model in WideResNet loss at 4000 labels.

- **Mean Teacher with DenseNet and Shake-Shake26**: We deemed it essential to test the

state-of-the-art SSML Mean Teacher model with a core network other than Shake-Shake26, hence the choice of DenseNet-121. Figure 20 shows that the training loss for both student networks is predictably lower, as the teacher network guides them using the EMA formula. These outcomes suggest that the teacher's parameters are optimized for the network's best overall performance. Moreover, the loss from the teacher represents the most optimal value attainable.

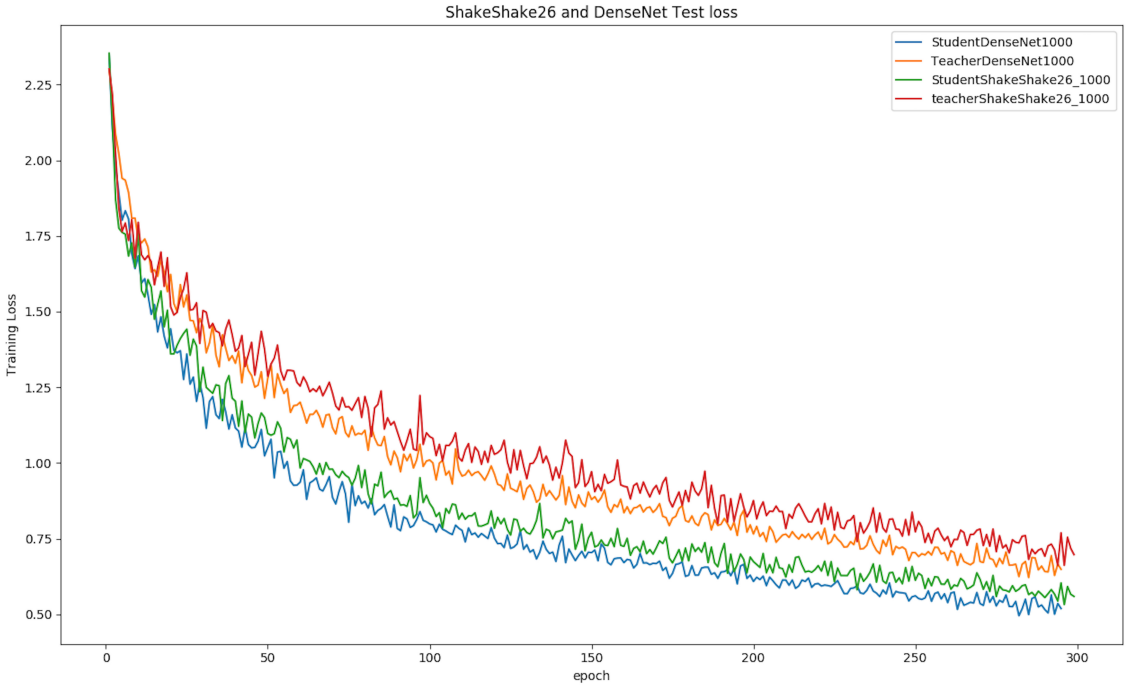

Figure 20: Student and teacher test loss in Mean Teacher for DenseNet and Shake-Shake26 at 1000 labels.

Additionally, the Shake-Shake26 network significantly outperforms DenseNet-121. As seen in Figure 21, this superiority is also reflected in accuracy performance. The primary reason is Shake-Shake26's broader and deeper network compared to DenseNet-121. However, it requires more computational time for training and hyperparameter tuning. System designers must consider this computational overhead, particularly when prioritizing performance over flexibility across datasets.

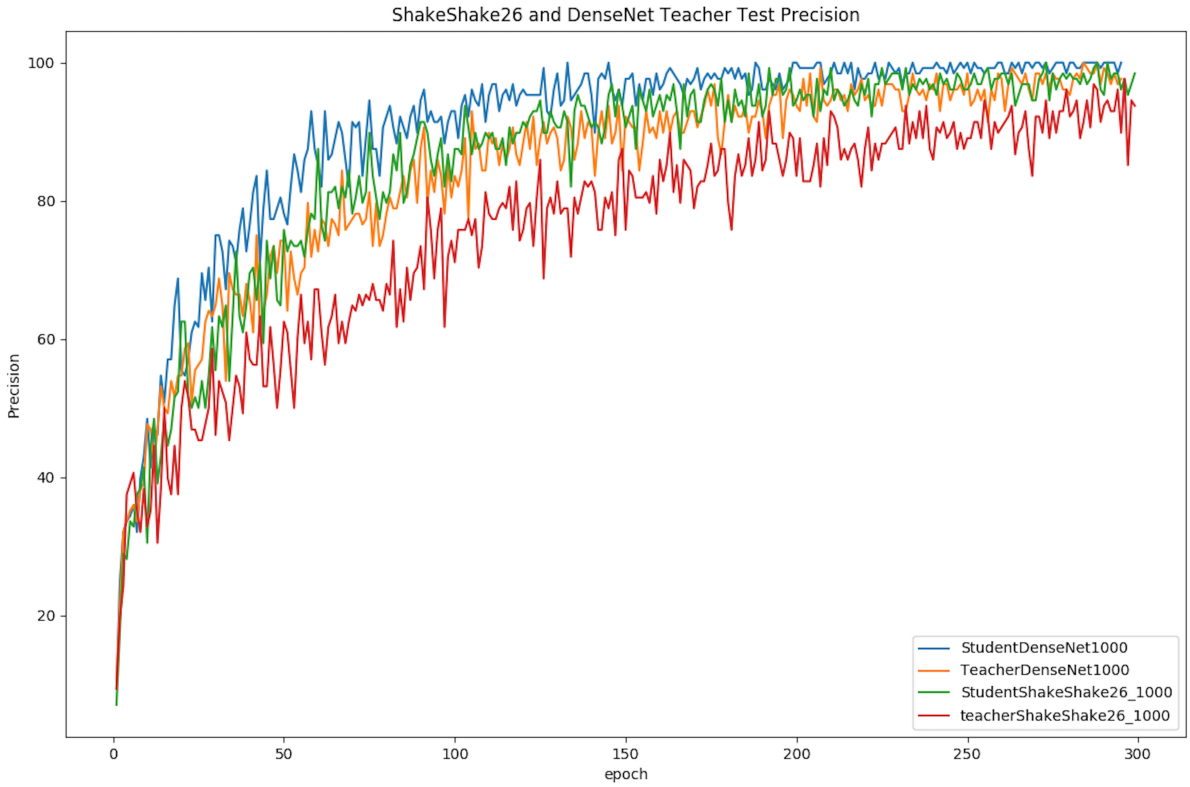

Figure 21: Student and teacher test accuracy in Mean Teacher for DenseNet and Shake-Shake26 at 1000 labels.

Figure 22 provides a few intriguing observations about the relationship between the teacher and the student models. First, with data with 4000 labels, the Shake-Shake26 network training process experiences a huge surge in the loss in the student nework. As a result, the teacher's network progress is also impacted. Secondly, the accuracy performance of a particular neural network is different for different datasets with various percentages of the labeled dataset. Therefore, it implies that the neural network design must be modified to incorporate dataset variations. Moreover, the hyperparameter tuning in the EMA can render better assumptions that would help in certain scenarios.

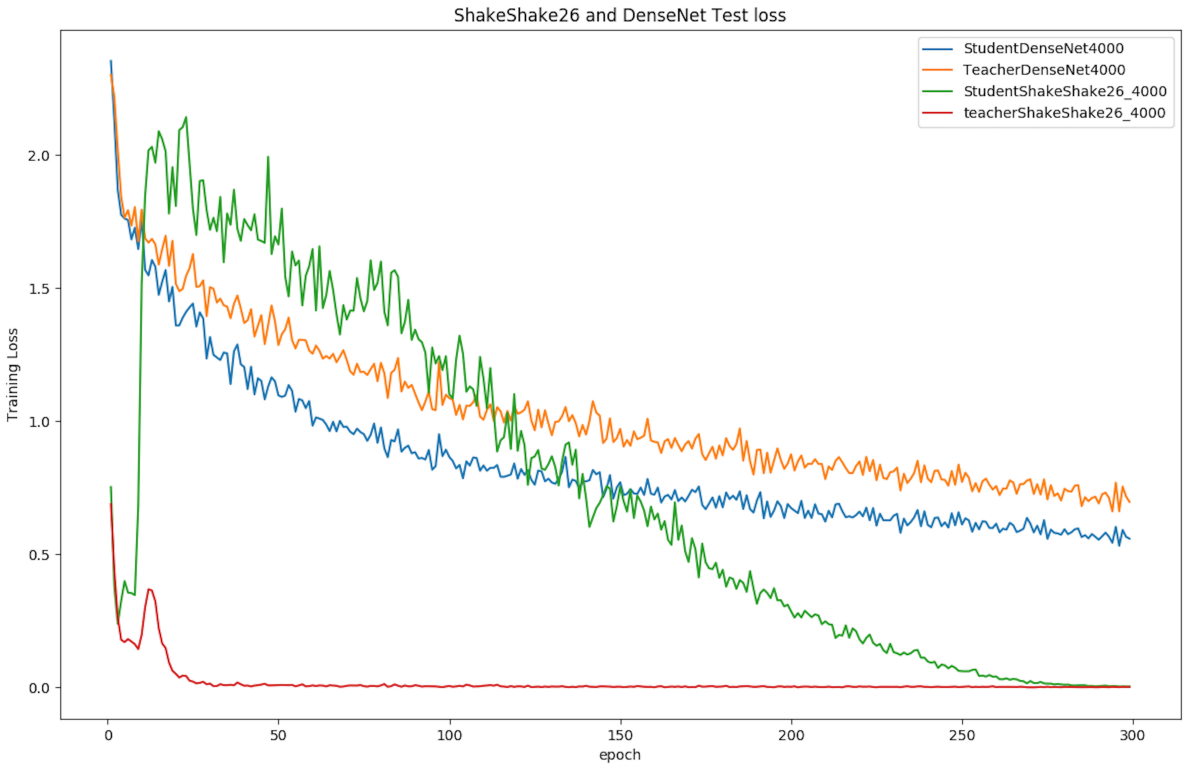

Figure 22: Shake-Shake26 vs. DenseNet-121 in Π model loss at 4000 labels.

Summary:

The incorporation of dual DNNs wthin a network classifier has shown to confer improved accuracy and reduced testing loss, a likely consequence of the combined classification prowess during forward propagation. This dual-network setup also injects additional variability into the calculation of the consistency cost, which in turn influences the total loss, with unlabeled data contributing to this effect. A comparative assessment of SSML models indicates that the Mean Teacher model surpasses the Π model in terms of accuracy and achieves a lower test loss.

When examining the performance across different SSML algorithms using Shake-Shake26, this model consistently presented a reduced loss when pitted against the DenseNet and WideResNet models, which may be attributable to its network width.

5.2.2 PF-SSCP with Different Network Classifiers

Understanding how PF-SSCP compares to CF-SSCP is crucial, but it is also vital to discern how the network classifier impacts PF-SSCP. We conducted tests using different network classifiers within

both the MixMatch and ReMixMatch frameworks. The subsections below detail our analysis.

Results of MixMatch and ReMixMatch Analysis:

- **MixMatch Compared to ReMixMatch with WideResNet as a Core Network**:

  Given PF-SSCP's emphasis on performance with the increased use of unlabeled data, we compared the outcomes from both frameworks using the same WideResNet-28-2, which has a depth of 28, a width of 2, and incorporates batch normalization [Ioffe and Szegedy, 2015]. As depicted in Figure 23, ReMixMatch slightly outperforms MixMatch with 4000 labels. This discrepancy widens with fewer labeled data.

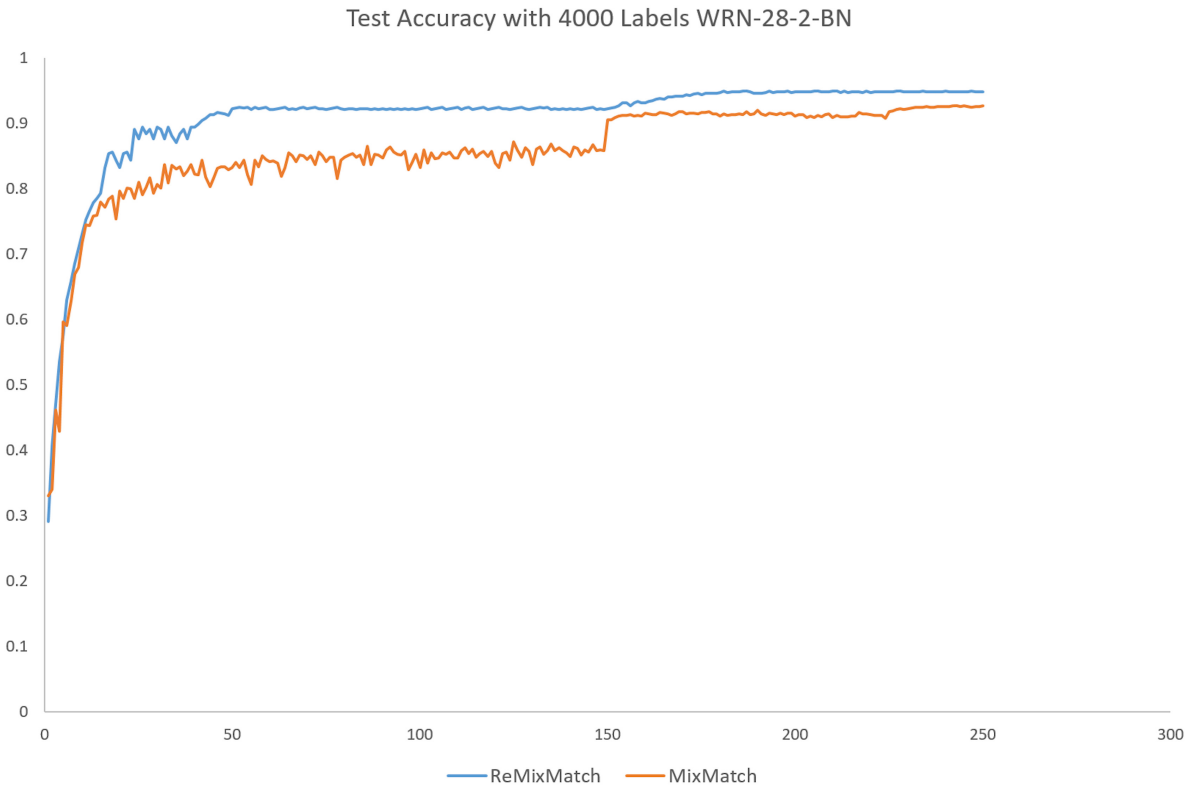

Figure 23: MixMatch vs. ReMixMatch at 4000 labels with WideResNet-28-2.

With 1000 labels, as shown in Figure 24, the performance gap becomes more pronounced. An intriguing observation is the convergence of the two SSMLs within a limited number of epochs. ReMixMatch achieves higher initial accuracy, indicating its strength with fewer labeled data. However, overall performance differences are marginal, as MixMatch eventually catches up by the end of training.

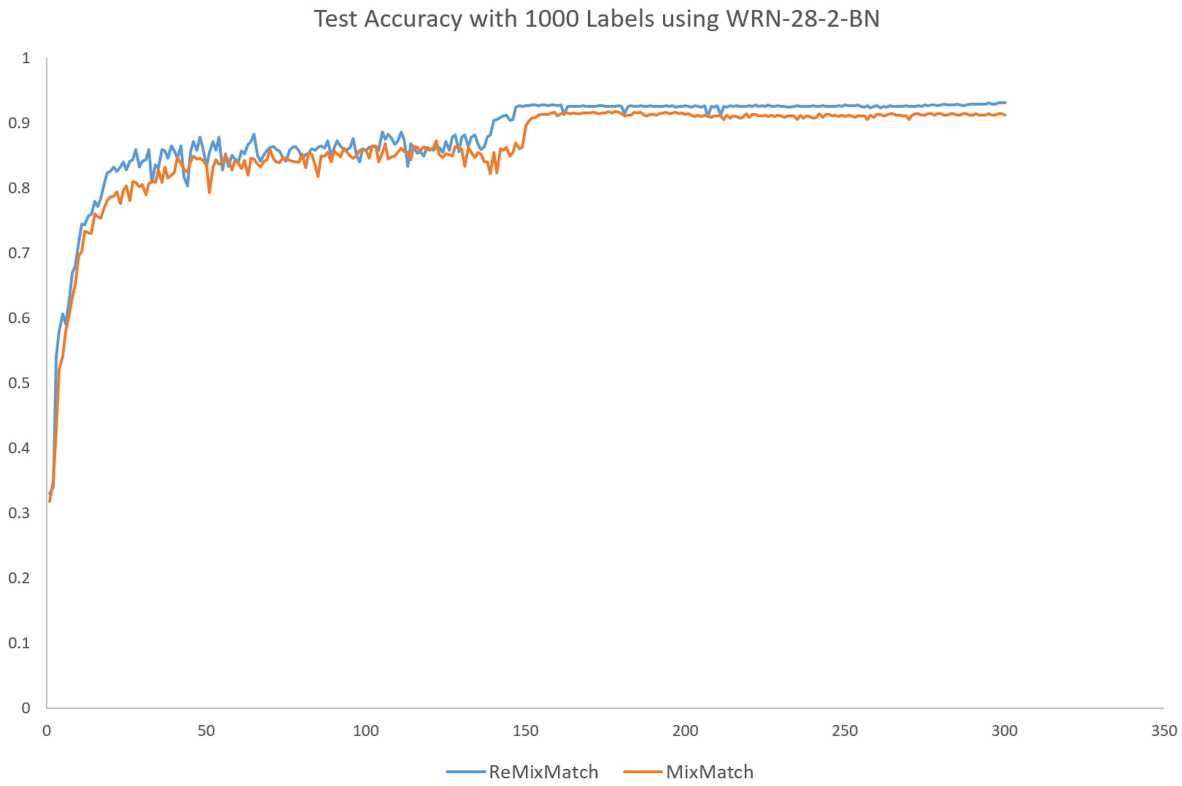

Figure 24: MixMatch vs. ReMixMatch at 1000 labels with WideResNet-28-2.

- **MixMatch and ReMixMatch with Shake-Shake26 as a core network**:

  Employing different core networks allows us to visualize the performance contributions of the SSMLs, independent of the underlying networks. By switching to Shake-Shake26, we sought to discern any performance variations. A close examination with 1000 labels, as seen in Figure 25, reveals an insignificant difference compared to WideResNet-28-2. Shake-Shake26 converges more rapidly, with marginally better accuracy, but this is not as pronounced when these DNNs are used independently.

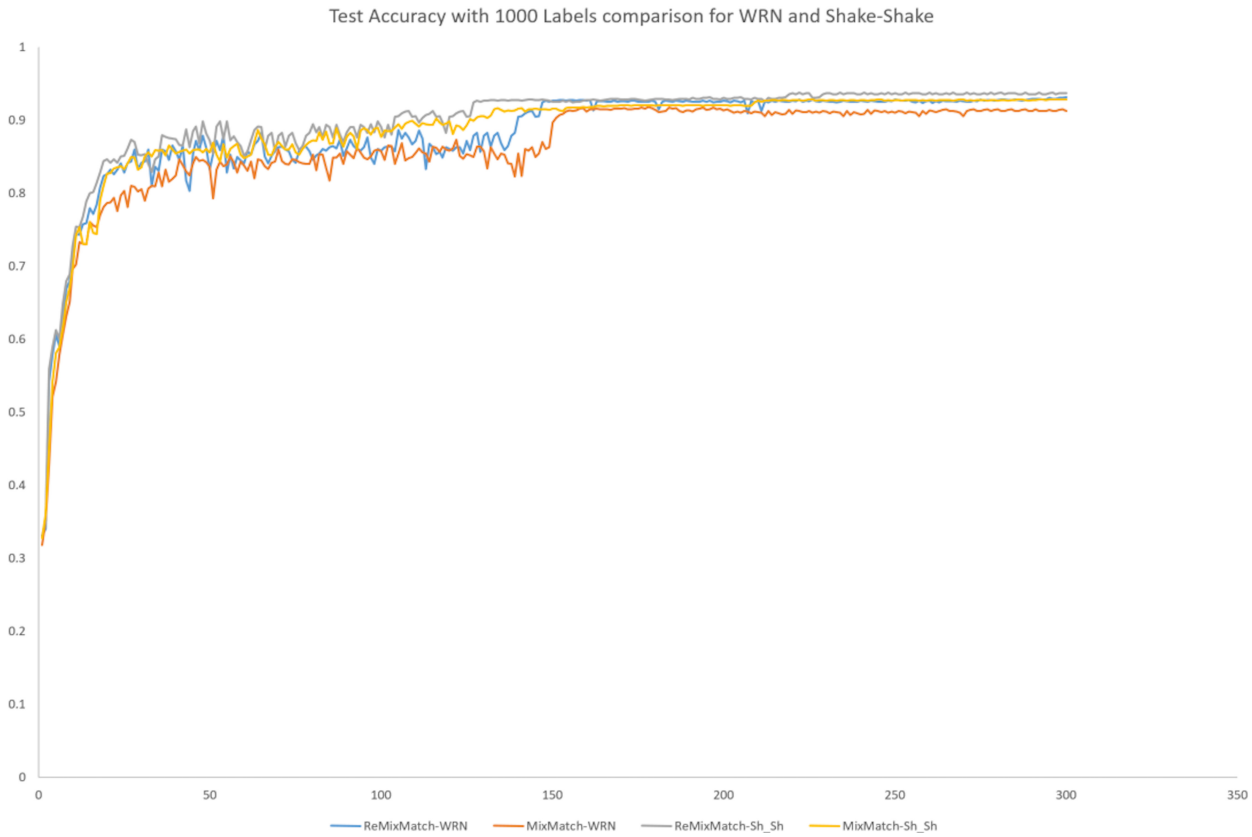

Figure 25: MixMatch and ReMixMatch at 1000 labels with WideResNet-28-2 and Shake-Shake26.

In Figure 26, with 4000 labels, ReMixMatch paired with Shake-Shake26 shows the highest accuracy. Notably, ReMixMatch with WideResNet and MixMatch with Shake-Shake26 are closely matched. The accuracy of ReMixMatch with WideResNet is slightly higher, but only by a narrow margin. Another key observation is the divergence in convergence patterns between MixMatch with Shake-Shake26 and MixMatch with WideResNet, as confirmed through five consecutive tests to ensure the finding's accuracy.

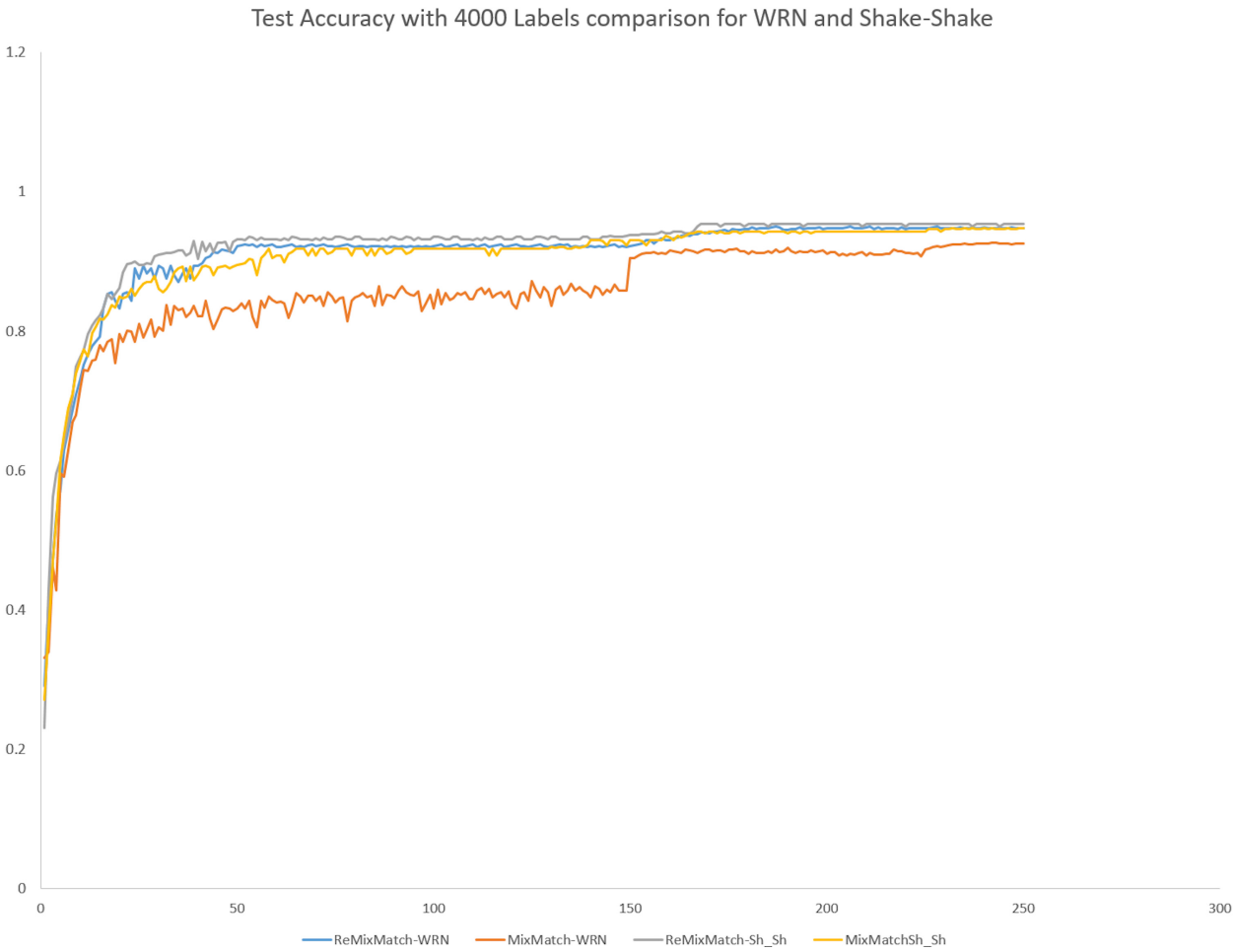

Figure 26: MixMatch and ReMixMatch at 4000 labels with WideResNet-28-2 and Shake-Shake26.

Summary: A comparative review of the experimental data reveals that MixMatch and ReMixMatch algorithms yield higher accuracy improvements over the Mean Teacher, Π model, and Temporal Ensembling methods, underscoring the significance of preprocessing in model performance. When juxtaposing MixMatch with ReMixMatch, the latter exhibits a marginal lead in accuracy, indicating its slight edge within the preprocessing-enhanced learning approaches.

5.2.3 SSML Training Time

Since the training time of a neural network in a production environment is critical, choosing the correct DNN for the SSML is a major decision in some cases where data change. Table 6 presents the training time results for every combination of all DNNs by executing all the SSMLs on the GPU—i.e., an Nvidia 2080 Ti GTX. The deeper and wider DNN is Shake-Shake26, and for this

reason, the results demonstrate the highest training time in this case. On the other hand, the lowest training time was observed for DenseNet-121. This is due to the interconnection among the layers [He et al., 2016a], making the total number of parameters lower as compared to the WideResNet model.

Table 6: Pearson correlation coefficients.

Network Classifier	SSML Algorithm	Labeled Data/ Unlabeled Data	Training Time (min)	PCC	Average PCC
Shake-Shake26	Temporal E.	49,000/1000 46,000/4000 0/50,000	1020 824 120	$PCC_{shake} = 0.7227$ p-value$= 0.002003$ $z'_{shake} = 0.929$	$PCC_{average} = 0.701$ p-value$= 0.003597$ $z'_{avg} = 0.87$
	Mean Teacher	49,000/1000 46,000/4000 0/50,000	1161 1094 158		
	Π model	49,000/1000 46,000/4000 0/50,000	**1351** 1272 168		
	MixMatch	49,000/1000 46,000/4000 0/50,000	1211 1103 163		
	ReMixMatch	49,000/1000 46,000/4000 0/50,000	1239 1142 149		
DenseNet-121	Temporal E.	49,000/1000 46,000/4000 0/50,000	323 276 **110**	$\boldsymbol{PCC_{densenet} = 0.7585}$ $\boldsymbol{p}$**-value$= 0.001046$** $\boldsymbol{z'_{densenet} = 0.993}$	
	Mean Teacher	49,000/1000 46,000/4000 0/50,000	392 337 124		
	Π model	49,000/1000 46,000/4000 0/50,000	443 407 168		
	MixMatch	49,000/1000 46,000/4000 0/50,000	404 353 131		
	ReMixMatch	49,000/1000 46,000/4000 0/50,000	411 367 139		
WideResNet	Temporal E.	49,000/1000 46,000/4000 0/50,000	387 324 132	$PCC_{wide} = 0.5968$ p-value$= 0.01904$ $z'_{wide} = 0.688$	
	Mean Teacher	49,000/1000 46,000/4000 0/50,000	537 485 158		
	Π model	49,000/1000 46,000/4000 0/50,000	553 512 179		
	MixMatch	49,000/1000 46000/4000 0/50,000	551 512 149		
	ReMixMatch	49,000/1000 46,000/4000 0/50,000	567 517 151		

Furthermore, in case of the Π model, the training time is higher due to the model's ability to perform backward propagation for both the student and the teacher models. Expectedly, the lowest training time is still that of Temporal Ensembling, as it includes only one DNN in the classifier,

which reduces the total number of parameters and the SSML's complexity.

Adding the PF-SSCPs, MixMatch and ReMixMatch require more training time as compared to the CF-SSCPs, excluding the Π model. This is the due to the complexity of the preprocessing step introduced and the pseudo-label calculation to generate the labeled and unlabeled training sets. Comparing MixMatch and ReMixMatch, it is clear that ReMixMatch requires more training time due to its more complex calculation of the pseudo-labels, as shown in Figure 7.

In order to see if the correlation would still withhold, Table 6 presents the Pearson correlation coefficients (PCC) [Vapnik, 1995] and the training time for all network classifiers independent of the SSCP with different SSML algorithms.

As shown previously, the training times for the Shake-Shake26 model as the DNN with a 49,000/1000 ratio of unlabeled/labeled data are shown as 1020, 1161, 1351, 1211, and 1239 min using the Temporal Ensembling model, Mean Teacher model, Π model, MixMatch, and ReMixMatch, respectively. On the other hand, the training times are 120, 158, 168, 163 and 149 min when there are no labels. In Table 6, we have similar results for the other network classifiers. The Pearson correlation coefficients (PCC) for each network classifier are used to indicate a positive correlation between the training time and the labeled/unlabeled data ratio.

Looking at PCC_{shake}, we can observe a correlation of 0.7227 with a p-value of 0.002003, compared to utilizing only the CF-SSCP algorithm with a p-value of 0.0278 and PCC_{shake} of 0.7227. Moreover, we can observe that $0.73 > 0.7227 > 0.7$ and $0.002003 < 0.0278 < 0.05$, which indicates a higher correlation. We can find similar results for the DenseNet model with a $PCC_{DenseNet}$ value of 0.7585. However, for WideResNet, we see a weaker correlation, although it is noticeably positive with a $PCC_{WideResNet}$ of 0.5968. To evaluate the average correlation coefficient among all three network classifiers, we used Fisher's Z to transform each correlation coefficient. Subsequently, we calculated the mean z' value in each case. This means z'_{avg} value was then transformed to the correlation coefficient again. After performing these computations, the PCC_{avg} was found to be 0.701, which is greater than 0.692, as stated with CF-SSCP, which indicates a higher correlation.

5.3 Summary of SSML Results and Addressing the Research Questions

As observed in the experiments, SSML based on PF-SSCP demonstrates higher accuracy and lower loss compared to that based on CF-SSCP, directly informing our conclusion regarding **RQ1.1**. This improved performance can be attributed to the quantity of data inputted into the DNNs, which plays a pivotal role in an SSML's effectiveness. This is due to the utilization of parameter updates in each design. Additionally, for both PF-SSCP and CF-SSCP network classifiers, we observed that fewer skip connections enhanced accuracy and diminished loss. This diversification of parameter updates facilitated more effective forward propagation, influencing the consistency cost.

Another insight gleaned from these experiments pertains to the correlation between training time and the ratio of labeled to unlabeled data, as addressed in **RQ1.2**. The presence of more labeled data necessitated increased computational time to calculate the loss, primarily due to the classification cost. Alterations in the classification cost significantly affected the loss, prompting substantial parameter updates during backward propagation, which, in turn, impacted subsequent forward propagation.

6 Generalizing the Methodology to SSL

Building on the previous section, this section describes the development of a UMAC process for SSL. It explains how the UMAC process was designed to enhance transparency and interpretability and presents the results of applying this methodology to SSL algorithms.

6.1 Generate SSL Model's Specific Computational Processes

In this section, we delve deep into individual state-of-the-art self-supervised methods. For each, we will dissect its unique architectural choices, breaking down its preprocessing strategies and network classifier design to understand the underpinnings of its performance.

6.1.1 MoCo Computation Process

Given an image I, we first perform data augmentation (random cropping, color jittering, and horizontal flipping) to produce two augmented views: I_q (query) and I_k (key).

$$I \rightarrow (I_q, I_k)$$

Both I_q and I_k are then processed through encoders. Specifically, f_q serves as the primary encoder for queries, while f_k operates as the momentum-updated encoder for keys.

$$q = f_q(I_q)$$
$$k = f_k(I_k)$$

The next step involves computing the InfoNCE loss [van den Oord et al., 2018] between the query and the positive key, amidst the backdrop of other negative keys sourced from the queue.

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k^+/\tau)}{\exp(q \cdot k^+/\tau) + \sum_{i=1}^{K} \exp(q \cdot k_i^-/\tau)}$$

where

- q is the query representation.
- k^+ is the positive key representation.
- k_i^- is negative key representations from the queue.
- τ is the temperature parameter.
- K is the number of negative samples in the queue.

Following the computation of the contrastive loss [Vapnik, 1995], the current image's key representation k is enqueued, and the oldest key representation is removed to preserve the queue's designated size. This methodology is visually represented in the computational process depicted in Figure 27. It is imperative to note, as observed in Figure 27, that the encoders f_q and f_k are symmetrical in architecture. This symmetry is foundational to the MoCo approach [He et al., 2020],

ensuring that both the query and key representations are generated from analogous structural bases. However, their update mechanisms differ, with f_q being directly updated through backpropagation and f_k relying on momentum updates from f_q. Turning our attention to parameter updates, we have the following:

- The parameters of the encoder f_q, denoted as θ, are updated directly using backpropagation based on the newly computed contrastive loss.
- Conversely, the parameters of the momentum encoder f_k, represented by ξ, are not updated through direct backpropagation. Instead, they are adjusted as an exponential moving average [Tarvainen and Valpola, 2017b] of the main encoder f_q's parameters.

$$\xi \leftarrow m\xi + (1-m)\theta$$

where

- ξ stands for the parameters of f_k.
- θ denotes the parameters of f_q.
- m is the momentum coefficient.

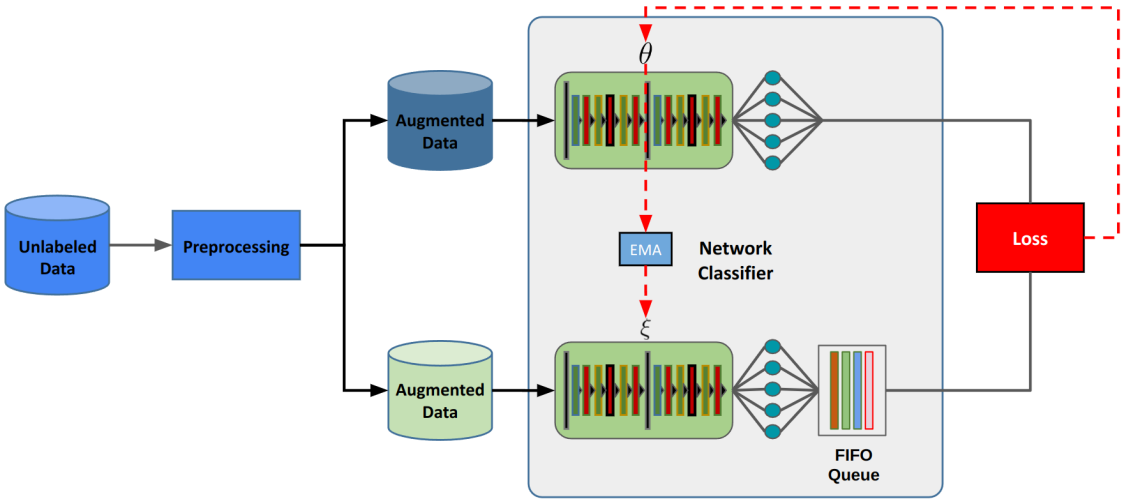

Figure 27: Computational process for MoCo.

6.1.2 MoCov2 Computation Process

Building upon the foundation laid out in Section 6.1.1 for MoCo, MoCov2 [Chen et al., 2020e] as shown in Figure 28 can succinctly be described as an evolved version with the following distinctive attributes:

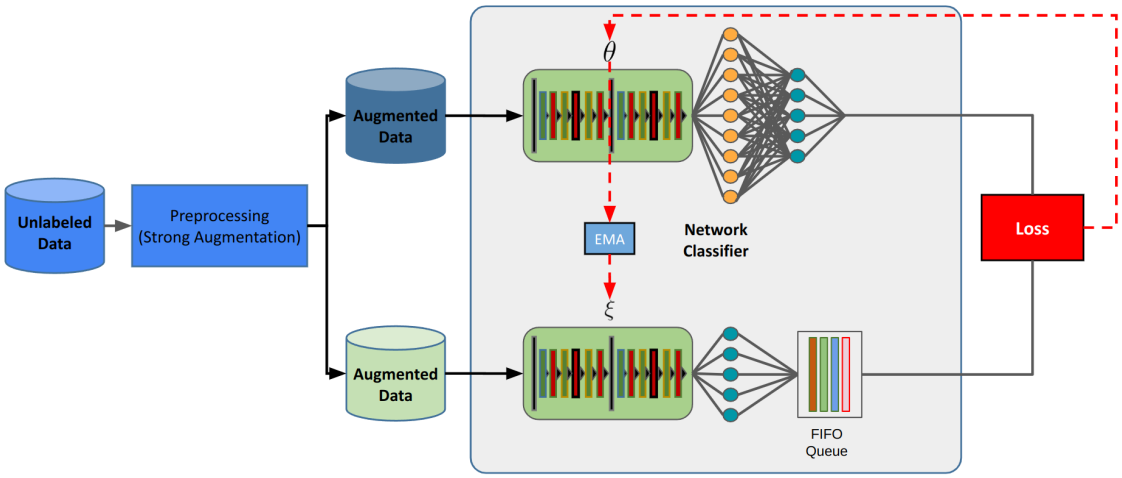

Figure 28: Computation process for MoCov2.

- Augmentation strategy: MoCov2 incorporates richer augmentations, including cropping (with flipping), color jittering, and Gaussian blurring.
- MLP head: An addition of a two-layer perceptron MLPs head provides a more expressive feature transformation.
- Batch normalization [Ioffe and Szegedy, 2015] absence in MLP: The MLP head in MoCov2 omits batch normalization, a choice empirically found to be beneficial.
- Learning rate schedule: MoCov2 uses a cosine learning rate schedule, eliminating the warm-up phase present in MoCo.
- Initialization strategy: MoCov2 directly initializes the momentum encoder with the main encoder's weights.

Collectively, these modifications propel MoCov2 to better performance in various tasks, accentuating its progress beyond MoCo.

6.1.3 SimCLR and SimCLRv2 Computation Process

SimCLR accentuates the significance of contrastive learning with larger batch sizes, where the quality and diversity of the negative samples (dissimilar pairs) play an indispensable role in model performance, as shown in Figure 29. Its foundation is the maximization of agreement between differently augmented views of the same data example through a learned representation [Chen et al., 2020b].

Unveiling SimCLR: Augmentation and Loss in Contrastive Learning

The cornerstone of SimCLR is its augmentation strategy. By utilizing combinations of random cropping, random horizontal flipping, color distortions, and Gaussian blurring, SimCLR increases the diversity of positive pairs, facilitating richer representation learning.

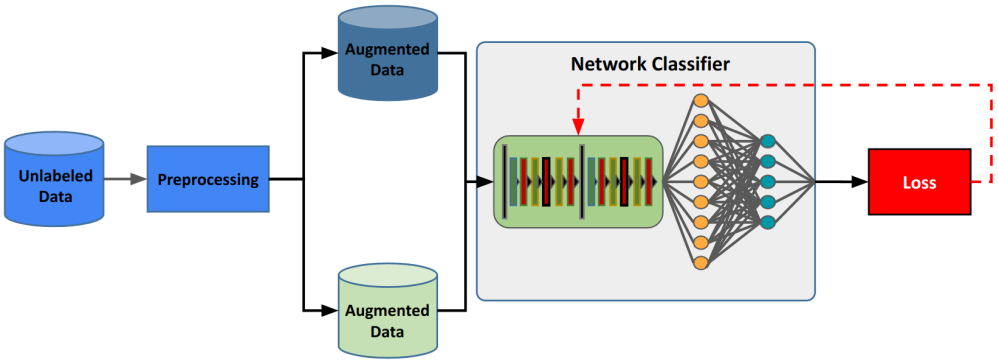

Figure 29: Computation process for SimCLR.

Its loss function, termed normalized temperature-scaled cross-entropy loss (NT-Xent), pivots on distinguishing between positive (similar) and negative (dissimilar) pairs. Representations of augmented versions of the same image are encouraged to be closer to each other in the embedding space than to other images. Mathematically, for a pair of representations x_i and x_j,

$$\text{NT-Xent}(x_i, x_j) = -\log \frac{\exp(\text{sim}(x_i, x_j)/\tau)}{\sum_{k=1}^{2N} \exp(\text{sim}(x_i, x_k)/\tau)}$$

where

- $\text{sim}(x, y)$ represents the similarity, computed as the dot product of $l2$-normalized vectors.
- N stands for batch size.

- τ is a pivotal temperature parameter, influencing the distribution's softness.

The temperature parameter τ demands careful selection as overly high or low values can push the model towards trivial solutions or impede convergence, respectively.

Interestingly, SimCLR employs a projection head, akin to MoCov2's MLP head, for its representation learning. This projection head, while functionally similar, is branded distinctively within the framework of SimCLR.

Advancing Contrastive Learning: SimCLRv2 Enhancements

SimCLRv2, building on the foundational principles of SimCLR, introduces a set of refinements that elevate its capabilities and performance. Opting for a more substantial architecture, SimCLRv2 incorporates the deeper ResNet-152, showcasing an empirical advantage over the conventional ResNet-50 [Chen et al., 2020c]. Further enhancing the pre-training process, a four-layer MLP projection head is employed; yet, distinguishingly, only the output of the base ResNet is harnessed as the representation for subsequent tasks. A pivotal strategy change lies in the fine-tuning process. Contrary to its predecessor, SimCLRv2 proposes fine-tuning the entirety of the model, encompassing the projection head, on the downstream tasks. This approach fosters a more integrative model refinement, leveraging the learned representations in a comprehensive manner.

Additionally, while the essence of SimCLR remains unsupervised, SimCLRv2 integrates supervised contrastive learning during its fine-tuning phase. This synthesis allows the model to utilize label information, ensuring enhanced class separation in the embedding space, and thereby potentiating the performance in classification tasks.

6.1.4 BYOL Computation Process

BYOL presents a paradigm shift in the self-supervised learning arena, as shown in Figure 30. Distinctively, it bypasses the need for negative samples in the contrastive loss formulation [Grill et al., 2020]. The architecture revolves around two neural networks: the target and the online networks. These entities evolve synchronously but exhibit different adaptation velocities.

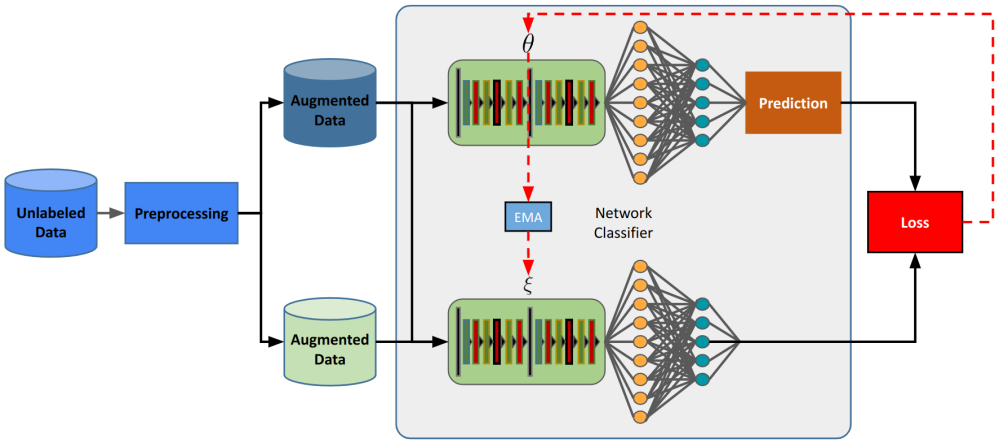

Figure 30: Computation process for BYOL.

The heart of BYOL is undeniably its predictor network, materialized as a multi-layer perceptron (MLP). This MLP, positioned subsequent to the primary encoding phase, transforms the image representation, amplifying the model's capability to grasp and interpret detailed patterns. Such an approach augments the model's efficacy without necessitating any alteration to the main encoder.

Parallelly, the EMA plays an indispensable role in BYOL. It orchestrates the parameter updates for the target encoder. While the main encoder witnesses continuous adaptations through backpropagation, the target encoder—sometimes dubbed the momentum encoder—undergoes updates rooted in the EMA of the main encoder's parameters. This methodology ushers in stability in the learning trajectory, ensuring a consistent evolution of the target.

In terms of loss computation, BYOL employs a symmetrized contrastive loss. The primary goal is to reduce the distance between two different views of an image, where one serves as an anchor in the online network and the other transits through the target network. The intent is to draw the predictor network's output (originating from the online network) and the target network's output closer in the representational space, optimizing their congruence.

A hallmark of BYOL is its conscious avoidance of negative sample utilization during loss computation. This innovative strategy deviates from classical contrastive learning paradigms, offering a streamlined learning objective.

6.2 Generating the Unified Model-Agnostic Computation for SSL

Inspired by our comprehensive scrutiny of contemporary techniques, our objective is to delineate a universal computational process. This process synergistically amalgamates the merits and pioneering strides of each methodology. As depicted in Figure 34, the integration of these methodologies forms a cohesive framework. The ensuing discourse spotlights the various components' virtues, elucidating the performance enhancements observed in each model attributable to specific design choices. With this context, it becomes imperative to raise a fundamental inquiry:

RQ4: How do encoder architecture, network configurations, auxiliary structures, and training strategies impact the model's performance in self-supervised learning?

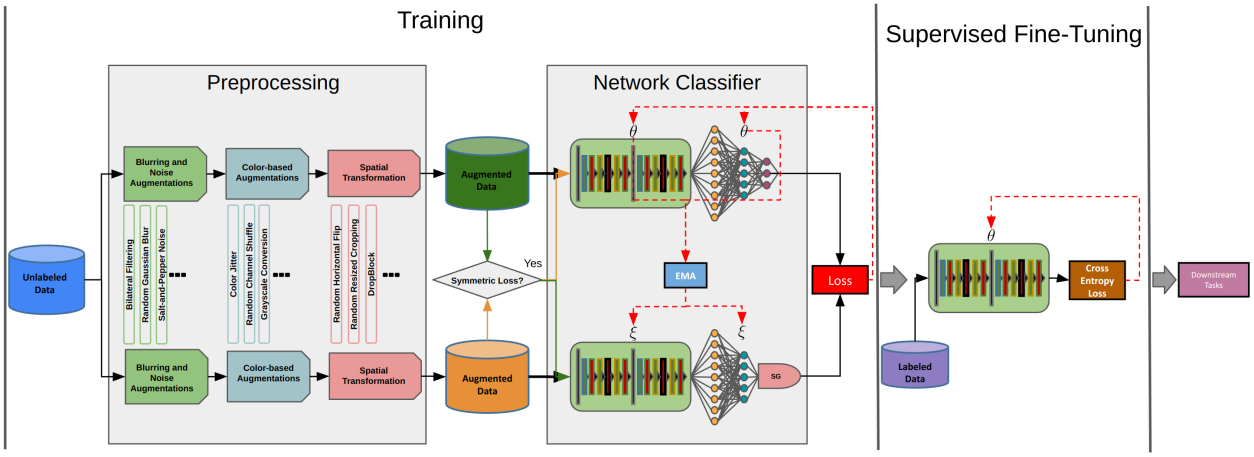

Figure 31: UMAC for self-supervised learning.

6.2.1 Training

The training phase is a critical component of our system's development, laying the foundation for a robust computational model. This stage is meticulously designed to enhance model performance through various preprocessing and network classification strategies.

- Preprocessing: Three distinct preprocessing strategies have emerged. To ensure each image rendition is unique, we employ a randomized approach, culminating in diversified data augmentation datasets:
  - Blurring and noise augmentation: Standard techniques employed to introduce subtle changes and perturbations to the input.

- Color-based augmentation: Introduces variations in the color spectrum of the input images.
- Spatial transformation: Meriting special emphasis, spatial transformations play a pivotal role in the augmentation strategy. The subsequent section elucidates the paramount importance of this technique, exploring multiple facets of its implementation.

- Network classifier: The prevailing models predominantly incorporate two encoders, punctuated with key components:
  - Encoder architecture: A concept that has withstood the test of time. The underlying principle is that grander architectures yield superior results, albeit at heightened computational costs. Incorporating specific components, especially the queuing of representations, can judiciously curtail the model's size and batch requisites without compromising performance.
  - Auxiliary components:
    * MLPs: Post-encoder MLPs are non-negotiable. A deeper MLPs on the key encoder relative to the queue encoder is essential. A more intricate and expansive encoder architecture mandates a correspondingly profound MLP. The adjoining table provides a pragmatic guideline.
    * Representation queue: The representation queue, while adhering to the FIFO principle, is also influenced by the learning rate of the key encoder. A higher learning rate necessitates a smaller queue. This is due to the fact that rapid weight updates in the encoder can swiftly render stored representations obsolete. Conversely, with a slower learning rate, the representations evolve more gradually, permitting a more extensive queue. Mathematically, the FIFO operation in terms of batches, influenced by a hyperparameter h can be articulated as

$$Q_{b+1} = \text{Append}(Q_b, k_{b+1}) - \text{Remove}(Q_b, h)$$

      where Q_b symbolizes the queue's state at batch b, k_{b+1} denotes the key representations of the newly processed batch, and h indicates the number of batch sizes'

worth of representations to be removed. Adjusting h allows for fine-tuning the refresh rate of the queue, providing a balance between queue longevity and representational freshness.

    * EMA: A straightforward procedure where solely the key encoder's value undergoes modifications, employing EMA to contemporaneously update the queue encoder's parameters.

- Loss function: The nature of the loss function plays a pivotal role in dictating the interaction between augmented data and the encoders, determining if both augmented datasets traverse both encoders or just one. Coupled with this, the role of queuing becomes evident:

  – Contrastive loss and queuing: In architectures that employ representation queuing, the contrastive loss is especially effective. A queue that captures representations from previous batches enables the network not only to contrast against the positive pair but also against a vast array of negatives. This extensive negative sampling sharpens the encoder's ability to discern between semantically close and diverse data points. In the absence of such a queue, the contrastive loss mainly depends on positive pairs, potentially overlooking the fine nuances provided by many negative samples. As such, leveraging the contrastive loss alongside a queue not only expands the range of representations but also enriches the learning process, setting a more comprehensive contrastive context.

  – Non-contrastive loss and queuing: For architectures employing a non-contrastive loss, there's a tendency to sidestep processing both augmented datasets through the two (or 'twin') encoders, choosing a more linear path. While this simplifies the computational trajectory, it might forgo the advantages of contrasting augmented views in a dense representational setting.

6.2.2 Supervised Fine-Tuning

The process of supervised fine-tuning fundamentally revolves around equipping the key encoder with capabilities to handle labeled data. At the heart of this process lies the widely adopted cross-entropy loss, which serves as the objective function for this phase of training.

Essentially, this entails a basic supervised training regimen for the key encoder. In contrast to the unsupervised or self-supervised paradigms previously discussed, here, the model explicitly learns from data that carry associated labels. Notably, only a percentage of the data, which is labeled, is employed for this fine-tuning. Often, this subset of labeled data is particularly used for benchmarking purposes to assess and compare model performances.

To facilitate the training, a softmax layer is appended at the tail end of the encoder. This layer's primary function is to produce probability distributions [Murphy, 2013] over the possible classes for each input sample.

Mathematically, if C denotes the number of classes, the output of the key encoder is fed into a softmax function adjusted to yield a C-dimensional vector. This vector essentially captures the likelihood of the input sample belonging to each of the C classes. The formula can be expressed as

$$\text{Softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$$

where x is the output of the key encoder and i ranges from 1 to C.

The cross-entropy loss, often used in classification tasks, measures the difference between the true labels and the predicted probability distributions. For a single sample, the cross-entropy loss $H(y, \hat{y})$ between the true label y and the predicted probability distribution $\hat{y}$ is given by

$$H(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$$

where C is the number of classes, y_c is the true label for class c (often a binary indicator of whether the sample belongs to class c or not), and $\hat{y}_c$ is the predicted probability for class c.

The produced probabilities are then contrasted with the true labels using the cross-entropy loss to guide the fine-tuning of the encoder. The loss is then backpropagated through the encoder to update its parameters.

Upon successful fine-tuning using the labeled data subset, the trained key encoder is subsequently utilized for various downstream tasks, harnessing its learned representations to tackle a broad spectrum of applications.

6.3 Experimental Design and Framework Specifications for SSL

Building on the comprehensive methodology outlined earlier, which encompasses the development of a UMAC process and its application through a structured two-step training and fine-tuning approach, we progressed to the empirical phase of our study. This phase was crucial for validating the theoretical underpinnings and practical efficacy of our proposed system. To this end, we strategically employed the CIFAR-10 dataset as the testing ground for our contrastive learning models. The CIFAR-10 dataset, renowned for its widespread use in visual recognition challenges, offered an optimal balance between complexity and manageability. This balance was particularly pertinent given our hardware capabilities, despite having access to a high-performance Nvidia GTX 4090 graphics card. Our choice was motivated by the desire to conduct a comprehensive array of model iterations and evaluations, ensuring thorough investigation within the bounds of our computational resources.

Our experimental strategy is primarily focused on addressing **RQ4**, which queries the effect of specific design decisions and the inherent strengths of our methodologies on model performance. To explore this, we devised four distinct experiments, each carefully crafted to illuminate the influence of different aspects of our UMAC framework and contrastive learning approach on performance metrics. Through methodical examination across these experimental conditions, our goal is to provide a comprehensive and empirically supported insight into the subtle interplays affecting model outcomes.

Experiment 1: SimCLR was put to the test, aligning closely with the original paper's methodologies. The experiment involved two main setups:

- SimCLR with basic augmentations:
  - Horizontal Flip (50% probability).
  - Resized Crop (32x32, scale: 0.8-1.0).
  - Color Jitter.
- SimCLR with enhanced augmentations: In addition to the above augmentations, spatial transformations like DropBlock and CutMix were incorporated.

This experiment primarily underscores the potency of augmentations, especially when the network classifier remains untweaked, mirroring its basic representation as depicted in the referenced figure.

Experiment 2: The focus of this experiment hinges on evaluating MoCo's performance using both symmetric and asymmetric losses, allowing a thorough investigation into the efficacy of the symmetric loss.

We have two primary setups for this analysis:

- *Asymmetric Loss:* This configuration strictly adheres to the original MoCo paper's methodology.
- *Symmetric Loss:* Diverging from the conventional MoCo approach, we introduce a symmetric loss. Here, in lieu of treating one crop as the query and the other as the key (as with the asymmetric loss), both crops play dual roles. After computing the loss using one configuration, the roles of the crops are interchanged, and an additional loss is calculated. This essentially emulates training for twice the number of epochs compared to its asymmetric counterpart. Such a practice not only aligns with strategies from SimCLR and BYOL but accentuates the iterative power of the dataset.

Preliminary observations suggest that symmetric loss often outperforms its asymmetric counterpart. One plausible justification is the enhanced dataset iteration, validating the adage: more epochs typically yield better results. Furthermore, this experiment unveils the potential of the EMA in a scenario where both the key and queue encoders process both sets of augmented images, especially under a symmetric loss framework.

Experiment 3: The essence of this experiment is to dissect the performance variations between MoCo and MoCov2. Specifically, we aim to decipher whether the advancements in MoCov2 are predominantly due to alterations in the augmentation techniques or the introduction of auxiliary components, like deeper MLPs n the network classifier. Our experimental setups are as follows:

- *Auxiliary Components Emphasis:* Here, we configure MoCov2 with the same augmentation techniques as the original MoCo to maintain a consistent augmentation baseline. The distin-

guishing factor is the modification in the network classifier of MoCov2 — the introduction of deeper MLPs as auxiliary components on the key encoder.

- *Augmentation Variance:* In this setup, we upgrade MoCo with the augmentation techniques originally designed for MoCov2. This helps discern how much of the performance enhancement in MoCov2 is attributable to its novel augmentation techniques.

Through this analytical approach, we aim to elucidate the relative contributions of advanced augmentation techniques and the introduction of auxiliary components in achieving the notable performance increments observed in MoCov2.

Experiment 4: The crux of this experiment revolves around examining the prowess of Supervised Fine-tuning across different state-of-the-art self-supervised learning models: SimCLR, MoCo, MoCov2, among others. We systematically vary the volume of labeled data to include only 1%, 2%, and 5% to understand the capabilities and limitations of each model under minimal supervision. Our experimental setups are outlined as:

- *Gradual Supervision:* For each model, we perform supervised fine-tuning with varying percentages of labeled data: 0%, 1%, 2%, and 5%. This will shed light on how minimal labeled data can be leveraged for effective model fine-tuning.

- *Full Supervised Training Comparison:* Additionally, we juxtapose the performance of the models when trained in a purely supervised manner with that of models that underwent supervised fine-tuning. This comparison aims to elucidate the true utility of self-supervised pre-training followed by supervised fine-tuning against pure supervised training.

Ultimately, through this investigative approach, our intent is to delineate the boundaries of effective supervised fine-tuning, especially when labeled data is scarce, and understand its comparative advantage over traditional supervised training paradigms.

6.4 Results and Comprehensive Analysis

To address the inquiries posed by **RQ4** regarding the influence of encoder architecture, network configurations, auxiliary structures, and training strategies on the efficacy of self-supervised learn-

ing models, our experimental phase was carefully structured. This phase, underpinned by the UMAC framework, was designed to dissect and evaluate the subtleties of various contrastive learning models, utilizing the CIFAR-10 dataset as a benchmark. Below, we present the outcomes of our experiments, meticulously linking each finding to the critical elements of **RQ4**, thereby offering a comprehensive perspective on how these diverse aspects collectively shape the performance landscape of self-supervised learning models.

6.4.1 Augmentation's Impact on SimCLR Performance

The experimental results from SimCLR, in alignment with the setups outlined, present some illuminating insights:

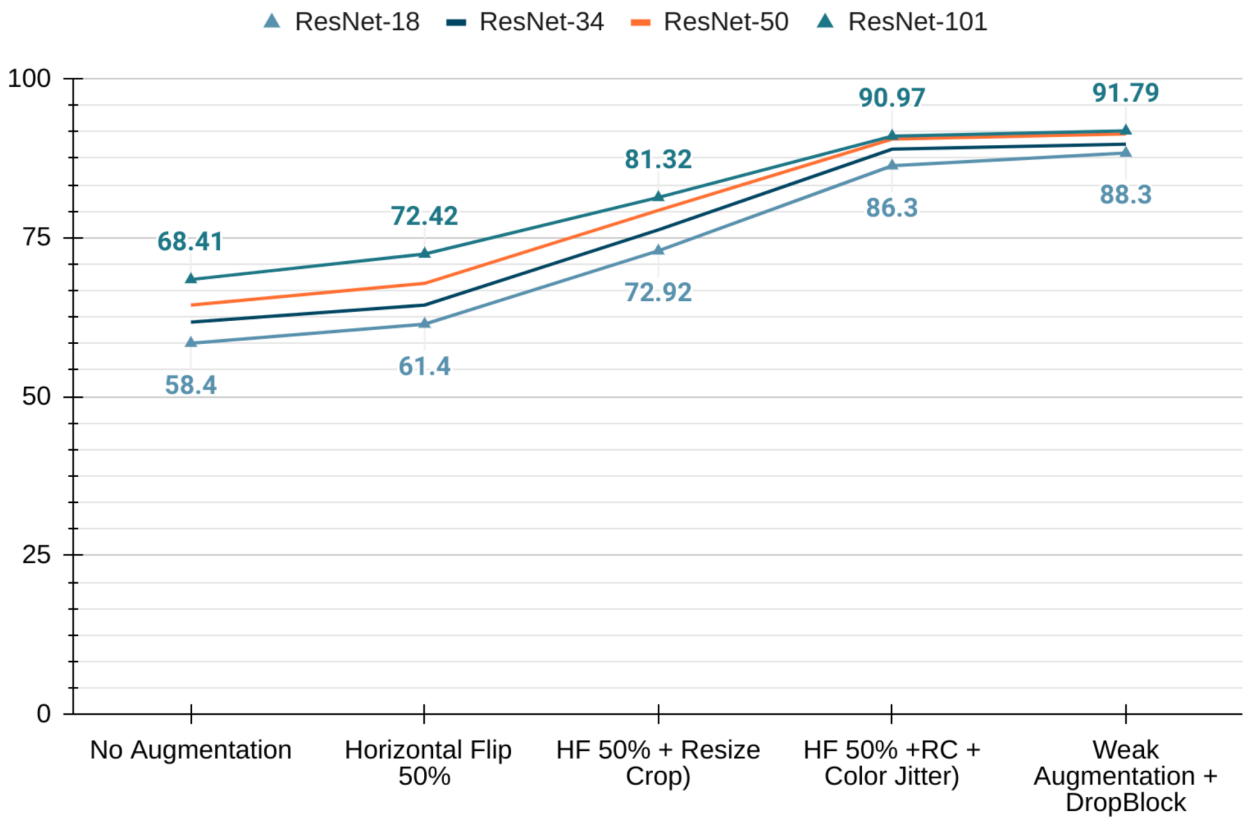

Figure 32: Top-1 Accuracy for SimCLR: Augmentations vs. Encoders (200 Epochs)

1. **Augmentation as a Performance Enhancer:** As illustrated in Figure 32, introducing additional augmentations noticeably boosts SimCLR's performance. This aligns with the understanding that data augmentation techniques are essential for self-supervised contrastive

learning, amplifying the model's capacity to discern distinct features and driving it to map semantically similar images closer in the embedding space.

2. **Network Size and Augmentation Synergy:** Augmenting the dataset with diverse and numerous transformations narrows the performance gap between varying ResNet architectures. While performance disparities between different-sized networks are minimized with enhanced augmentations, it remains evident that architectures with more parameters generally exhibit superior results. The interplay between comprehensive augmentations and network complexity underscores a balancing act: augmentations elevate the representational power of networks, yet architectural depth and complexity maintain their intrinsic advantage.

In summation, this experiment robustly affirms the notion that enriching the dataset with a broader range of augmentations can significantly bolster performance. This stands as a testament to the pivotal role that data augmentation plays in self-supervised learning landscapes, particularly within the SimCLR framework.

6.4.2 Performance Evaluation for Symmetric vs. Asymmetric Losses

We conducted a series of evaluations to discern the impact of symmetric and asymmetric losses in the MoCo training regime. The experiments were performed across different training lengths to ascertain if the benefits of symmetric loss consistently persist over extended epochs. Our observations, represented as Top-1 accuracy percentages, are presented in the table below:

Model-Loss	**200Ep.**	**400Ep.**	**800Ep.**
ResNet-18 Asymmetric	82.51	86.32	88.73
ResNet-18 Symmetric	85.35	88.53	89.74
ResNet-50 Asymmetric	85.23	87.4	89.52
ResNet-50 Symmetric	87.22	89.17	90.78

Table 7: Top-1 accuracy evaluation of ResNet architectures (combined with loss type) over varying epochs.

A deep dive into the results makes the prowess of symmetric loss vividly apparent. Throughout the training epochs, models employing symmetric loss continually outperformed those using the conventional asymmetric loss. This underscores the anticipated benefit we postulated in **Experiment 2**, emphasizing the iterative power of the symmetric loss configuration.

One noteworthy observation from Table 7 is the performance of ResNet-18 with symmetric loss. Even though ResNet-18 is intrinsically a smaller and potentially less expressive model than ResNet-50, when trained with symmetric loss, it not only narrows the performance gap but even surpasses the larger ResNet-50 model trained with asymmetric loss. This observation stands as a testament to the potency of symmetric loss in harnessing better representational capacities even from smaller architectures.

6.4.3 Comparative Efficacy: MoCo and MoCov2 in Light of Augmentations and Auxiliary Components

In conducting this experiment on the CIFAR-10 dataset, it essentially mirrors prior research but with a deviation: we forwent the inclusion of the cosine learning rate schedule. We postulated that the effects of the cosine learning rate schedule could be mitigated by intensifying the augmentation techniques, particularly through the introduction of both CutMix and DropBlock.

Case	Configurations			Encoder		
	MLP	A	A+C+D	R18	R34	R50
MoCo				82.5	82.9	83.13
(a)	✓			83.22	83.48	83.81
(b)		✓		83.82	84.12	85.14
(c)			✓	84.17	84.52	85.1
(d)	✓	✓		84.7	85.28	86.04
(e)	✓	✓	✓	85.1	86.12	87.78

Table 8: MoCo variants' performance at epoch 200, showing the effects of using an MLP head, standard and advanced augmentations (A, A+C+D), and different ResNet encoders (R18, R34, R50). Checkmarks (✓) indicate the applied configurations.

Our findings reaffirmed the significance of auxiliary components, notably the deeper MLP in MoCov2's network classifier. Upon analysis, it was evident that while the performance metrics achieved by both models were roughly equivalent, there was a marked increase in computational complexity with MoCov2, attributable to its deeper and more intricate network classifier.

However, it's crucial to acknowledge that the intensified augmentations did offer a substantial performance boost, implying that effective data augmentation can, to an extent, rival the enhancements brought about by network classifier modifications. This experiment has further solidified the belief that while auxiliary components and intricate network designs have their merits, augmenta-

tions can be a more cost-effective method (in terms of computational demands) to enhance model performance.

6.4.4 Evaluating Self-Supervised Models with Limited Labeled Data for Supervised Fine-Tuning

In **Experiment 4**, our exploration gravitated towards the intricacies of self-supervised learning. We aimed to understand how Supervised Fine-tuning plays out across a spectrum of leading self-supervised learning models, particularly when one is limited by the paucity of labeled data.

% of L.	**SimCLR**	**SimCLRv2**	**MoCov2**	**BYOL**
0%	86.3	87.4	86.7	86.5
1%	90.8	92.8	92.1	92.3
2%	91.8	94.2	93.3	93.8
5%	92.4	95.2	94.8	94.9

Table 9: Performance dynamics of various self-supervised learning models using ResNet-50 as the encoder under different magnitudes of labeled data. (L.: = labeled data percentage)

The data presented in Table 9 unveils several compelling narratives. First and foremost, the ascendancy of SimCLRv2 stands out. As evident from the table, SimCLRv2's performance, especially in low data label scenarios, surpasses its counterparts. Its holistic design, which integrates the entire UMAC (detailed in section 6.2), offers it this distinctive edge.

However, pivoting our gaze from the table reveals another contender deserving accolades - BYOL. Even though the table might suggest SimCLRv2's supremacy, a deeper delve into BYOL's metrics vis-a-vis its resource efficiency paints a different picture. For settings where computational bandwidth is constrained, BYOL's ability to deliver remarkable outcomes with less overhead suggests it might be an optimal choice.

Beyond these model-specific insights, Table 9 presents a macroscopic revelation. The performance metrics, largely oscillating around the early 90s percentile, are a testament to the potency of combining self-supervised pre-training with sporadic supervised fine-tuning. This set of experiments underscores the prevailing belief: when juxtaposed against traditional supervised training paradigms, this novel approach stands tall.

To cap it off, while the discussion might oscillate between champions like SimCLRv2 and BYOL due to their stellar numbers, the underlying message remains steadfast: in scenarios where

labeled data is a precious rarity, self-supervised models emerge as a robust alternative.

6.5 Summary of SSL Results and Addressing the Research Questions

This subsection distills key findings from our experiments, particularly those gleaned through the UMAC in subsection 6.2. Our framework demystifies the complexities of self-supervised learning, highlighting the essential architectural and strategic components that drive performance and interpretability. These components, critical in the realm of XAI, collectively enhance our understanding of model mechanics, a topic we will delve into with our upcoming analysis.

As we proceed, we'll directly address our research questions, shedding light on the intricate interdependencies among these core components and their cumulative impact on model outcomes.

6.5.1 Unified Computation Process: Key Components and Strategies

Our evaluation of self-supervised learning models has led to the identification of key components that significantly influence their performance. These elements form the cornerstone of our proposed UMAC Framework, which is aligned with the principles of XAI and underscores the importance of transparency and clarity in algorithmic processes.

RQ2 Summary: The primary components integral to a unified computation process in the evaluation of self-supervised learning models encompass the encoder architecture, network configurations, auxiliary structures, and training strategies. These factors are central to the effectiveness of our UMAC, which advocates for greater transparency and clarity in algorithmic processes, consistent with XAI principles.

Following our examination of these primary components, it's evident that a nuanced approach to the encoder architecture is necessary, one that considers the specific demands of the tasks at hand. Similarly, network configurations must not only be robust but also adaptable, capable of accommodating the diverse and dynamic nature of various learning scenarios. These strategic considerations underscore the complexity of developing effective self-supervised learning systems and highlight the need for a multifaceted strategy.

RQ3 Summary: Improving performance and interpretability in self-supervised learning requires the fine-tuning of encoder architectures to meet specific task requirements, optimizing network configurations for improved efficiency, integrating auxiliary structures that enhance interpretability, and adapting training strategies to ensure better convergence and generalization. These adaptations contribute to creating models that are not only more effective but also more comprehensible, thereby broadening their potential applications.

The implications of these findings are profound, suggesting that the path to more effective and interpretable self-supervised learning models lies not just in the sophistication of the models themselves, but in a holistic approach to their development and training. This encompasses everything from the initial architecture design to the final training phases, demanding a consistent focus on transparency and adaptability at every stage.

A deep understanding of these components, the functionality of auxiliary components, and the choice of loss functions is indispensable for the development of versatile and interpretable self-supervised learning models.

6.5.2 Impact Analysis of Key Factors in Self-Supervised Learning

Our experiments have demonstrated the profound impact of encoder architectures, network configurations, auxiliary structures, and training strategies on the performance of self-supervised learning models. While complex architectures like ResNet-50 tend to outperform more straightforward models, this advantage is subject to modulation by a variety of elements, suggesting a nuanced interplay among these factors.

The configuration of the network and the incorporation of auxiliary components have a marked effect on learning efficacy. Strategies such as the implementation of deeper MLPs and symmetric loss functions have been observed to facilitate the extraction of richer representations. It is important to note, however, that these improvements generally come at the cost of increased computational demands.

In addition, our research has shown that strategic augmentation strategies can substantially enhance performance, in some cases more so than modifications to network complexity, thus providing a more cost-effective approach.

Importantly, models pre-trained through self-supervision and subsequently fine-tuned with lim-

ited labeled data have exhibited robustness, underscoring the adaptability and efficiency of these methods in environments where data is scarce.

RQ4 Summary: The performance of self-supervised learning models is significantly influenced by a combination of factors, including the complexity of the encoder, network configurations, auxiliary structures, and training strategies. Complex encoders generally yield superior performance, but the effectiveness of augmentation strategies and loss functions also play a key role in modulating performance. While modifications to the network can improve learning outcomes, they also tend to increase computational demands. Notably, self-supervised models demonstrate considerable robustness, even when fine-tuned with limited data, highlighting their potential for wide-ranging applicability.

7 Applying UMAC to Design a Deep Learning Model in Medical Image Classification

Deep learning methods have made significant strides in assisting clinicians with rapid examination and accurate diagnosis [Litjens et al., 2017]. However, these methods often demand large amounts of data, which is typically scarce in the medical field. Factors like limited patient data or inadequate access to medical equipment can lead to biased and overfitting models [Litjens et al., 2017]. While data augmentation is a common solution, it requires specialized knowledge for medical image modalities (e.g., MRI, CT, X-ray) and is computationally expensive [Shorten and Khoshgoftaar, 2019, Perez and Wang, 2017]. Standard image-level augmentation methods often fall short in enhancing sample diversity or achieving meaningful semantic transformations, while generative models, though useful for improving diversity, remain complex and resource-intensive [Cubuk et al., 2019, Ratner et al., 2017].

Recent advancements in feature-level data augmentation, like the implicit data augmentation method ISDA, offer an alternative by generating new data within the feature space [Wang et al., 2019]. These methods apply computationally efficient techniques such as random disturbances, interpolations, or extrapolations. However, they largely focus on semantic direction without adequately addressing semantic strength, leading to potential label violations. Additionally, generative models [Goodfellow et al., 2014, Antipov et al., 2017, Shin et al., 2016, Frid-Adar et al., 2018],

while beneficial, require complex training processes and significant computational resources. Automated augmentation methods [Cubuk et al., 2019, Ratner et al., 2017], though less dependent on manual tuning, are still computationally intensive. Feature-level augmentation techniques [DeVries and Taylor, 2017, Zhang et al., 2017, Yun et al., 2019b, Hendrycks et al., 2019] operate in the deep feature space but often miss the balance between semantic direction and strength.

Building on our prior work [Neghawi and Liu, 2024] with the UMAC framework, which was originally designed to unify and enhance self-supervised learning, we now extend UMAC to address the challenges in medical image classification. The model-agnostic nature of UMAC allows it to adapt to different learning paradigms and environments, making it particularly effective in handling the scarcity of medical data, while meeting the high demands for accuracy and interpretability. By integrating XAI principles, UMAC helps develop models that are both transparent and reliable, offering potential for significant advancements in the medical field.

7.1 Challenges in Acquiring Medical Data

Obtaining high-quality, labeled data is a significant challenge in the medical field, particularly for training machine learning models. Unlike other domains where data can be easily generated or collected, medical data acquisition is often constrained by several unique factors.

First, data privacy and security are major concerns. Medical data contains sensitive information, and regulations such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States and the General Data Protection Regulation (GDPR) in Europe impose strict controls on data collection, storage, and sharing. These regulations are essential for protecting patient privacy but can also limit access to large datasets for research purposes [Shen et al., 2017, McDermott et al., 2021].

Additionally, data labeling and annotation is a complex process in the medical domain. It requires expertise from trained medical professionals, which makes it time-consuming and expensive. The variability in labeling due to subjective interpretations by different experts adds another layer of complexity [Irvin et al., 2019, Oakden-Rayner, 2020].

Another challenge is data diversity and representation. Many medical datasets lack diversity, both in terms of demographic representation and the range of medical conditions covered. Most

available datasets are sourced from a limited number of healthcare institutions, often in high-income countries, leading to biases that affect the generalizability of models to broader or underserved populations [Kaushal et al., 2020, Roberts et al., 2021].

Furthermore, data size and quality can be limiting factors. Certain medical conditions are rare, making it difficult to obtain sufficient data to train robust models. The quality of available data may also vary significantly due to differences in imaging equipment, protocols, and patient populations, leading to noisy and inconsistent datasets that hinder model performance and reliability [Esteva et al., 2017, Willemink et al., 2020].

The use of medical data for research raises ethical and legal challenges, particularly around consent and data misuse. Researchers must navigate complex ethical guidelines to ensure data usage aligns with patient rights and expectations, requiring careful collaboration among clinicians, ethicists, and data scientists [Cirillo et al., 2020, Vayena et al., 2018].

Finally, data integration and standardization pose significant difficulties. Integrating data from multiple sources, such as electronic health records (EHRs), imaging, and genomic data, requires significant preprocessing, cleaning, and normalization to ensure compatibility and meaningful analysis [Raghupathi and Raghupathi, 2014, Johnson et al., 2016].

These challenges highlight the need for innovative approaches like the UMAC framework, which aims to maximize the utility of available data through advanced computational techniques and data augmentation strategies. By addressing the limitations of current datasets, UMAC can improve the performance and generalizability of machine learning models in the medical field.

7.2 Motivation for UMAC Adoption in Medical Image Classification

The UMAC framework addresses critical challenges in medical image analysis, where labeled data is often scarce, and model performance is heavily reliant on appropriate data augmentation. UMAC offers a structured and controlled approach to augmentation, ensuring that meaningful data transformations are applied, which is crucial for a variety of downstream tasks.

7.2.1 Downstream Tasks Benefiting from UMAC

UMAC is particularly beneficial in tasks such as segmentation, classification, and anomaly detection, where the availability of labeled data is limited. By narrowing down augmentation choices and ensuring that the data remains diagnostically meaningful, UMAC helps enhance model performance across these tasks:

- **Segmentation**: Precise segmentation of tumor boundaries or regions of interest is critical in medical diagnosis. UMAC enhances segmentation models by applying controlled augmentations that mimic real-world variabilities, preserving key medical features while increasing data diversity.

- **Classification**: In classification tasks, such as disease detection and severity grading, UMAC ensures that models generalize well to unseen data by applying the most suitable augmentation techniques. This helps create realistic variations in imaging conditions and patient characteristics, improving model accuracy.

- **Anomaly Detection**: UMAC supports anomaly detection by selecting augmentations that simulate rare or underrepresented cases, ensuring that models are exposed to a wider range of conditions without introducing unrealistic data distortions.

7.2.2 Challenges Without UMAC

Developing machine learning models in the medical field is complex and costly, particularly due to challenges in selecting and applying augmentation techniques. UMAC addresses these challenges by:

- **Data Scarcity and Imbalance**: Medical datasets are often small and imbalanced, leading to overfitting and poor generalization. UMAC simplifies the selection of augmentation techniques and controls the amount of augmentation applied, creating meaningful data diversity while preventing overfitting.

- **Controlled Augmentation**: Choosing the right augmentation techniques is critical, as excessive transformations can compromise the diagnostic value of medical images. UMAC

sets algorithmic limits on the level of augmentation to ensure that key features of the data are preserved, maintaining the integrity of the images.

- **Model Generalization**: Generalizing models to different patient populations and imaging conditions is a significant challenge. UMAC provides a structured vision for applying augmentations that reflect real-world medical variability, allowing models to perform well across diverse scenarios while avoiding the introduction of distortions that could affect prediction accuracy.

7.3 Augmentation Strategies for Medical Image Data in the Context of the UMAC Framework

The challenges of acquiring and processing medical data highlight the need for advanced data augmentation techniques, particularly in the context of semi-supervised and self-supervised learning, where labeled datasets are often scarce. Several state-of-the-art (SOTA) augmentation strategies have been developed to improve the utility and diversity of medical image datasets, enhancing model robustness while addressing data scarcity and imbalance. Notable among these techniques are GuidedMixup, PuzzleMix, ReMix, ResizeMix, and SaliencyMix [Zhang et al., 2021c, Kim et al., 2020, Chou et al., 2020a, Uddin et al., 2021].

7.3.1 Implicit Semantic Data Augmentation (ISDA)

ISDA operates in the latent semantic space of medical image data, generating augmented examples based on the underlying features of the images, such as variations in tumor shape, size, or texture. This approach produces meaningful augmentations that stay true to the medical data distribution, allowing models to handle rare and complex conditions more effectively.

By modeling the latent semantic features, ISDA generates realistic augmentations that maintain the essential characteristics of the original data, providing improved generalization for medical image classification tasks [Wang et al., 2020].

7.3.2 Bayesian Semantic Data Augmentation (BSDA)

BSDA introduces a probabilistic framework for augmenting medical image datasets, particularly those suffering from class imbalances. It leverages Bayesian inference to simulate synthetic examples for underrepresented conditions, creating more diverse and representative datasets. This technique is especially valuable for rare medical conditions where collecting sufficient real-world data is challenging.

By generating synthetic data points that reflect the probabilistic distribution of relevant features, BSDA ensures that augmented examples are realistic and representative of medical conditions. This helps improve model performance by providing variability in the training data while maintaining the integrity of the original dataset [Zhao et al., 2021].

7.3.3 Integrating and Leveraging BSDA and ISDA Techniques for Augmentation in UMAC

The UMAC framework integrates the advanced augmentation techniques from both BSDA and ISDA, building upon their unique capabilities to handle data scarcity and imbalance in medical image datasets. Below, we outline how each technique contributes to the augmentation strategy within UMAC, and how they are combined for optimal performance in medical machine learning.

- **Feature-Level Augmentations from BSDA**:
  - **Synthetic Data Generation**: By sampling probabilistic distributions of underrepresented conditions, UMAC generates synthetic data points that reflect rare medical conditions (e.g., specific anomalies or tumors) to enhance dataset diversity.
  - **Class-Conditional Augmentation**: This technique ensures that rare or minority classes are specifically targeted with augmentations, such as controlled interpolation of features, improving dataset balance without violating class semantics.
- **Semantic Augmentations from ISDA**:
  - **Latent Semantic Transformations**: UMAC applies augmentations that mirror variations in medical conditions, such as tumor shape and size, generating semantically meaningful data without distorting the core medical features .

– **Random Perturbations**: Small perturbations in the feature space create new data variations while preserving the semantic coherence of the medical images, helping to improve model generalization .

By leveraging these feature-level and semantic augmentations, UMAC goes beyond traditional augmentation methods to generate diverse, semantically rich data. The combination of BSDA's probabilistic framework and ISDA's latent semantic transformations allows UMAC to handle the challenges of small, imbalanced medical datasets effectively.

7.3.4 UMAC's Adaptive Augmentation Workflow

The flexibility of the UMAC framework allows it to integrate multiple augmentation techniques from both BSDA and ISDA, while also incorporating custom augmentations tailored to the specific medical imaging task. This adaptive approach ensures that augmentations are applied contextually, depending on the scarcity of data, the complexity of the medical condition, and the demands for model generalization.

The augmentation workflow in UMAC is broken into distinct phases, where each phase applies a specific augmentation strategy designed to increase data diversity while maintaining the medical relevance of the images. Figure 33 provides a visual overview of how these limitations and constraints are applied during the augmentation process.

- **Phase 1: Blurring and Noise Augmentation** simulates real-world imaging imperfections and variability. This phase introduces Gaussian blur and low-intensity noise to the images, mimicking noise found in real-world medical imaging devices (e.g., MRI or X-ray machines). The level of blur and noise is carefully controlled using metrics like the Structural Similarity Index (SSIM) to ensure that diagnostic features are not obscured. The SSIM is calculated as:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{14}$$

  Here, μ_x and μ_y are the average pixel values of the original and augmented images, σ_x^2 and σ_y^2 are the variances, and σ_{xy} is the covariance between the images. The constants C_1 and

C_2 are small values to avoid division by zero. A threshold of SSIM ≥ 0.9 ensures that the blurring and noise augmentations do not significantly degrade the image quality.

- **Phase 2: Color-Based Augmentation** introduces variations in brightness, contrast, and hue to ensure generalization across images captured using different equipment or lighting conditions. The key challenge in medical imaging is that small changes in contrast or brightness can alter the visibility of critical features. Hence, in this phase, color-based transformations are constrained by the Mean Absolute Error (MAE) between the original and augmented images, calculated as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i| \tag{15}$$

Here, x_i and y_i are the pixel values of the original and augmented images, and n is the total number of pixels. By setting a threshold for MAE (e.g., MAE $\leq 5\%$), UMAC ensures that these color-based augmentations do not distort important medical features.

- **Phase 3: Spatial Transformations** applies rotations, scaling, and cropping to simulate positional variations, ensuring the model is invariant to changes in orientation or size. Medical images such as X-rays or MRIs can be taken from different angles or with varying magnification, so it's critical that these transformations preserve the structural integrity of the images. Rotations are applied at fixed intervals (90°, 180°, and 270°), and scaling is limited to a range of 90%-110% to avoid significant distortions. The impact of these transformations is measured in the feature space using Cosine Similarity, defined as:

$$\text{Cosine Similarity} = \frac{\mathbf{f_x} \cdot \mathbf{f_y}}{||\mathbf{f_x}|| \, ||\mathbf{f_y}||} \tag{16}$$

Here, $\mathbf{f_x}$ and $\mathbf{f_y}$ are the feature vectors of the original and augmented images, and $||\mathbf{f_x}||$ and $||\mathbf{f_y}||$ are their magnitudes. A threshold of Cosine Similarity ≥ 0.95 ensures that the spatial transformations do not lead to significant changes in the semantic meaning of the image.

These phases operate within UMAC's overarching architecture, where probabilistic sampling from BSDA and latent semantic transformations from ISDA guide augmentation choices. Specif-

ically, ISDA is used to ensure that feature-level perturbations are semantically meaningful and reflect plausible variations of medical conditions, such as changes in tumor size or shape.

By combining these augmentation techniques, UMAC enhances the diversity of training data, improves model robustness, and ensures that the deep learning models generalize effectively across different medical imaging conditions. Each augmentation is tightly controlled by similarity metrics like SSIM, MAE, and Cosine Similarity to maintain the diagnostic relevance of the images, making UMAC highly adaptable to the demands of medical image classification.

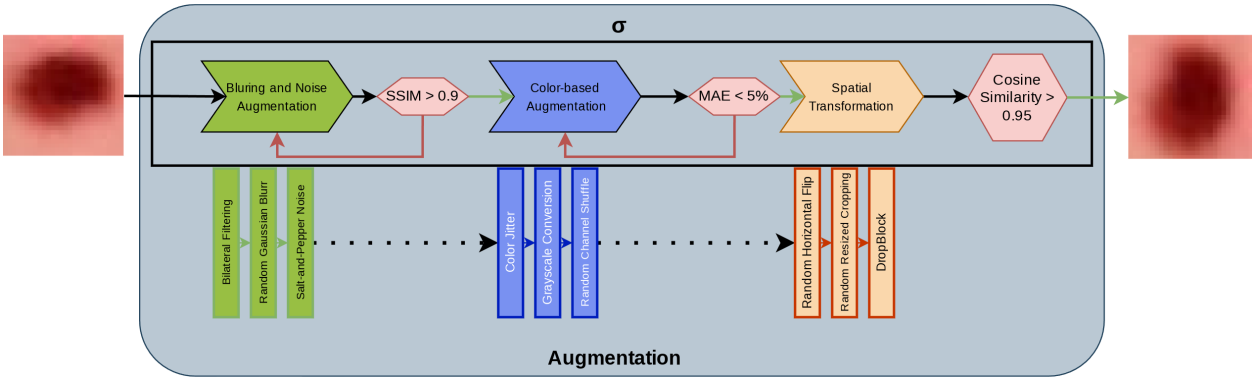

Figure 33: Workflow of Augmentation Techniques in the UMAC Framework: An Overview of How Augmentation Strategies Are Applied to Improve Model Robustness and Generalization in Medical Imaging.

7.4 UMAC Design for Medical Applications

The UMAC framework offers a structured design pattern analogous to software componentization and separation of concerns in software engineering. UMAC's modular design integrates SOTA augmentation techniques to ensure the highest learning performance in medical image analysis.

To adapt the UMAC methodology [Neghawi and Liu, 2024] to the medical domain, we first outline its core components and the steps required to develop a UMAC system. This development necessitates a structured approach that efficiently integrates a variety of computational models, algorithms, and frameworks, while remaining scalable. The goal is to build a computational system versatile enough to manage different data types, problems, and computational environments while also incorporating cutting-edge advancements in methodologies. The process can be summarized in four key steps:

- **Identify the State-of-the-Art (SOTA):** Research and identify the latest methodologies and best practices applicable to the problem. Our investigation began by focusing on SOTA methods in SSL, particularly in the context of medical image classification. The core difference we explored lies in the augmentation techniques tailored to medical data, where domain-specific challenges such as data scarcity and the need to preserve medical features are paramount. We identified two major SSL augmentation methods: BSDA and ISDA, both of which were analyzed in detail. These techniques were selected for their ability to address feature-level augmentation in a way that aligns with the complexity of medical images.

- **Analyze SOTA Solutions:** Test and assess each SOTA solution on various datasets to understand their capabilities and limitations. The evaluation of BSDA and ISDA was performed on multiple datasets, as shown in earlier sections. We systematically reviewed how these approaches enhance data diversity while preserving key semantic features in the images. BSDA was noted for its ability to synthesize realistic examples that reflect underrepresented conditions, addressing class imbalance. ISDA, on the other hand, provided strong latent feature augmentation, preserving essential medical image characteristics like tumor shape and size, thus preventing the degradation of diagnostic information. This analysis underscored the value of these methods for the medical domain, where overfitting and poor generalization due to limited data are common issues.

- **Design Computation Processes:** The existing SSL setups were leveraged as foundational frameworks for integrating these augmentations. BSDA and ISDA were central to developing the UMAC Augmentation for the medical field, providing a solid foundation for creating augmentations tailored to medical data. These techniques ensured that transformations reflected real-world medical variabilities while preserving critical diagnostic features. BSDA offered a structured method for generating synthetic, realistic examples, particularly for rare conditions, while ISDA enhanced augmentation by ensuring that alterations in the latent space remained semantically meaningful. Although BSDA and ISDA are distinct methods from UMAC, UMAC leverages the underlying principles of both approaches—focusing on preserving the semantic integrity of data and generating diverse, realistic augmentations. Together, these techniques formed the basis of the augmentation strategies integrated into

UMAC, specifically designed for medical image analysis. UMAC further accounted for the balance and limits of augmentation to avoid introducing artifacts that could mislead diagnostic outcomes.

- **Develop the UMAC System:** The final step was integrating all the components into the UMAC system, with a focus on augmentations within the SSL framework. BSDA and ISDA were seamlessly incorporated to dynamically adapt to different datasets. The system automatically selects and applies augmentation strategies, such as spatial transformations or noise augmentation, without compromising medical data integrity. This comprehensive UMAC setup ensures that models can generalize effectively across classification tasks while preserving diagnostic accuracy.

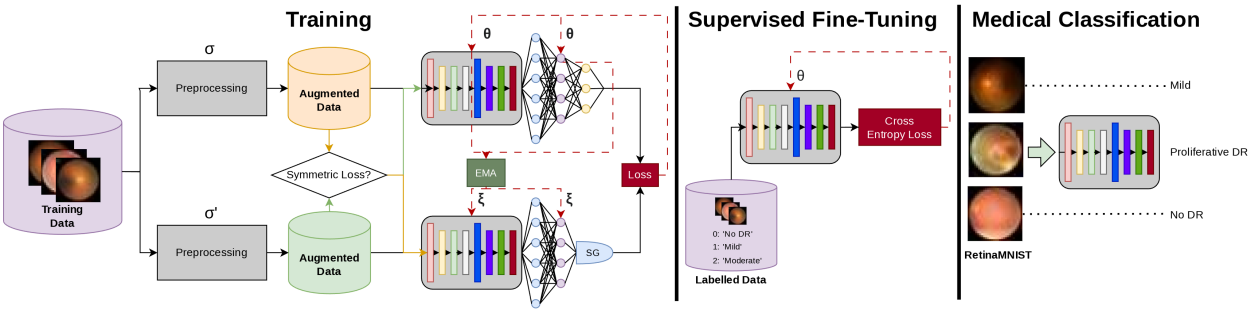

Figure 34: UMAC with SSL in the medical field. In this context, θ and ξ represent parameters, while σ and σ' refer to random parameters.

In applying UMAC to the medical domain, we place significant emphasis on preprocessing techniques, which play a critical role in boosting the performance of SSL models for medical image classification. Building on insights from our prior research [Neghawi and Liu, 2024], we have developed a systematic approach that unifies and abstracts computational models and algorithms to ensure their adaptability across various data types and computational environments. Specifically, we utilize the UMAC for SSL, concentrating on data augmentation techniques to enhance model performance in medical image classification, as depicted in Figure 34.

Figure 34 outlines the Unified Agnostic Computation Process for self-supervised learning as applied to medical image classification. The process begins by generating two augmented datasets from the original dataset, which are then processed through two encoders: the Key Encoder and the Queue Encoder. The Key Encoder is represented by parameters θ, while the Queue Encoder is

initialized with parameters ξ.

To ensure diversity in the dataset, random parameters σ and σ' are applied during augmentation. After the data passes through the encoders, contrastive loss is calculated by comparing positive pairs (similar images) with negative pairs (dissimilar images). A FIFO queue mechanism is employed to store representations from previous batches, allowing the model to learn from a broader range of examples. This enhances the model's ability to distinguish between different classes.

If a symmetric loss function is applied, the augmented input is fed into both network classifiers, which have different parameters: θ for the Key Encoder and ξ for the Queue Encoder. The loss is computed between the two classifiers, typically CNNs or transformers, to capture differences in representation.

Finally, the parameters of the Key Encoder (θ) are used as the starting point for the fine-tuning phase. The output is further fine-tuned using labeled data, which enhances the model's performance in tasks such as medical image classification, as demonstrated in the RetinaMNIST dataset.

7.4.1 Training

The training phase plays a crucial role in the development of our system, serving as the groundwork for building a strong computational model. This stage is carefully structured to improve model performance by employing diverse preprocessing techniques and network classification methods.

- **Preprocessing:** The detailed procedure for image data augmentation is outlined in Algorithm 2. This algorithm aims to increase the dataset size while preserving the original label distribution. Given an original set of images $X = \{x^1, x^2, x^3, \ldots, x^n\}$ and corresponding labels $Y = \{y^1, y^2, y^3, \ldots, y^n\}$, where each label $y^i \in \{0, 1, 2, \ldots, u\}$, the objective is to expand the dataset by a factor α.

  The algorithm executes the following steps:

  - Initialization: Calculate the target size of the augmented datasets as:

$$m = \lceil n \times \alpha \rceil$$

Algorithm 2 Image Data Augmentation with Size Increase

Require:
| Original set of images $X = \{x^1, x^2, x^3, \ldots, x^n\}$
| Original set of labels $Y = \{y^1, y^2, y^3, \ldots, y^n\}$, where $y^i \in \{0, 1, 2, \ldots, u\}$
| Desired augmentation factor α

Ensure: Augmented image sets S_1, S_2 with size increased by factor α

$S_1, S_2 \leftarrow \emptyset$
$m \leftarrow \lceil n \times \alpha \rceil$
$P(y) = \{p_0, p_1, \ldots, p_u\}$ from Y
$C(y) = \{c_0 = 0, c_1 = 0, \ldots, c_u = 0\}$
for $i = 1$ to 2 **do**
  for $j = 1$ to n **do**
    $\sigma \leftarrow$ rand()
    $B_i, N_i, C_i, O_i \leftarrow \sigma_i$
    $c_{ik} \leftarrow$ rand(), $\forall k \in [1, k_{\text{color}}]$
    $o_{ik} \leftarrow$ rand(), $\forall k \in [1, k_{\text{spatial}}]$
    $A_i \leftarrow$ createAugmentationFunction$(B_i, N_i, C_i, O_i, c_{i1}, \ldots, c_{ik}, o_{i1}, \ldots, o_{ik})$
    $X_{\text{aug}} \leftarrow A(x^j)$
    S_i.add(X_{aug})
  end for
  while size of $S_i < m$ **do**
    Select a random number $r \in \{1, 2, \ldots, n\}$
    Determine label $y_r \leftarrow y^r$
    if $C(y_r) < p_{y_r} \times m$ **then**
      Get image x^r from X
      $\sigma \leftarrow$ rand()
      $B_i, N_i, C_i, O_i \leftarrow \sigma_i$
      $c_{ik} \leftarrow$ rand(), $\forall k \in [1, k_{\text{color}}]$
      $o_{ik} \leftarrow$ rand(), $\forall k \in [1, k_{\text{spatial}}]$
      $A_i \leftarrow$ createAugmentationFunction$(B_i, N_i, C_i, O_i, c_{i1}, \ldots, c_{ik}, o_{i1}, \ldots, o_{ik})$
      $X_{\text{aug}} \leftarrow A(x^r)$
      S_i.add(X_{aug})
    end if
  end while
end for
return S_1, S_2

Then, initialize empty sets S_1 and S_2 for storing augmented images. The label distribution $P(y)$ is calculated from Y to ensure the original distribution is preserved in the augmented datasets.

– Augmentation Strategy: Each image in the dataset X undergoes random transforma-

tions to create augmented versions. The augmentation function A is applied using randomly generated parameters:

$$X_{\text{aug}} = A(x)$$

Every image is augmented at least once to ensure diversity across the augmented datasets.

- Maintaining Label Distribution: To keep the label distribution consistent with the original dataset, additional augmentations are performed selectively. Images are chosen at random, and their augmentation count $C(y)$ is tracked to match the original label proportions:

$$\text{If } C(y) < p_y \times m, \text{ augment image } x.$$

- **Final Datasets:** The final augmented datasets S_1 and S_2 reach the target size m, with label distributions closely matching those of the original dataset. This process helps the model generalize better and handle a broader range of data.

To further illustrate this augmentation process, Figure 35 provides a concrete example using a DermaMNIST image. This example demonstrates the series of transformations applied during augmentation, such as random color shifts and spatial adjustments, highlighting how these changes ensure diversity in the training data.

- **Network Classifier:** Most of the models employ two encoders, with several key components as outlined below:

  - **Encoder Architecture**: The encoder can be any of the popular CNN architectures such as ResNet [He et al., 2016b], ResNeXt [Xie et al., 2017], or DenseNet [Huang et al., 2017]. Larger architectures generally yield superior results, albeit at heightened computational costs. Specific components, especially the queuing of representations, can judiciously curtail the model's size and batch requisites without compromising performance. The minibatch size is set to 128 by default for each of these CNN architectures but can be adjusted if needed. However, we did not find significant differences

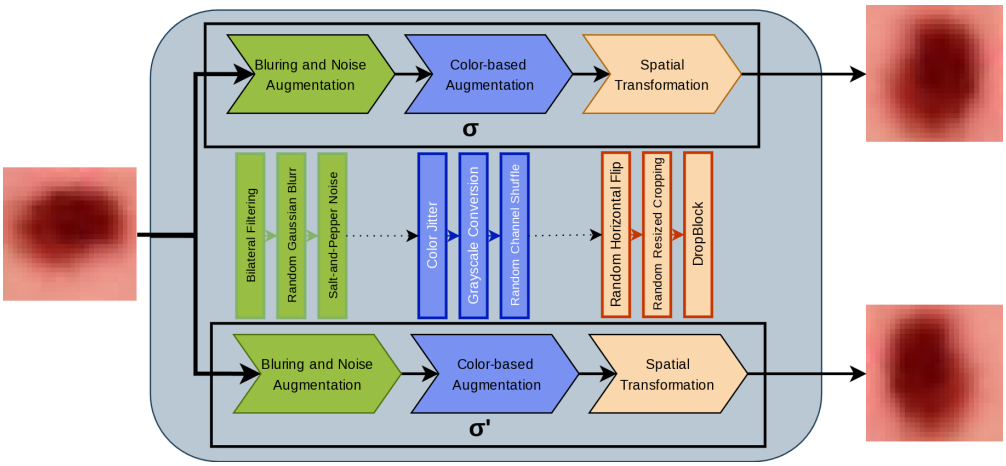

Figure 35: Example of Augmentation Function applied to a DermaMNIST image, showcasing color shifts and spatial transformations.

when altering the minibatch size, which is a limitation of our experiment. In addition to CNN-based encoders, Transformer-based architectures like Vision Transformer (ViT) [Dosovitskiy et al., 2020] or Swin Transformer [Liu et al., 2021b] can also be utilized for image classification tasks, providing versatile options for the encoders based on self-attention mechanisms.

– **Auxiliary Components**:

  * **MLPs**: Multi-layer perceptrons (MLPs) following the encoder are crucial. The Key Encoder benefits from a deeper MLP compared to the Queue Encoder, and as the Encoder Architecture becomes more complex, the MLP depth should increase accordingly.

  * **Representation Queue**: The queue, governed by a FIFO mechanism, is influenced by the learning rate of the key encoder. A higher learning rate calls for a shorter queue due to rapid weight updates that render old representations obsolete. A slower learning rate allows for a longer queue, as the representations evolve more gradually. The queue's operation in terms of batches, controlled by a hyperparameter h, can be expressed as:

$$Q_{b+1} = \text{Append}(Q_b, k_{b+1}) - \text{Remove}(Q_b, h)$$

Here, Q_b is the state of the queue at batch b, k_{b+1} represents the key representations of the new batch, and h indicates the number of batch sizes to be removed. Adjusting h allows for a balance between queue size and representation freshness.

* **EMA**: The Key Encoder's parameters are updated via exponential moving averages (EMA), which also updates the Queue Encoder in parallel.

Let θ represent the randomly initialized parameters of the Key Encoder, and ξ the parameters of the Queue Encoder. During training, the two augmented datasets are used, and the backward propagation updates θ, while ξ is updated using EMA:

$$\xi \leftarrow \beta\xi + (1 - \beta)\theta$$

where β is the EMA decay rate. This process is illustrated in Figure 34.

- **Loss Function:** The loss function plays a crucial role in determining how the encoders interact with the augmented data. It also influences whether the augmented datasets are processed by both encoders or just one. Additionally, the queuing mechanism becomes significant in the following ways:

  - **Contrastive Loss and Queuing:** In models using a representation queue, contrastive loss proves particularly effective. A queue that stores representations from previous batches allows the model to contrast the positive pair against a large number of negative samples. This enhances the model's ability to differentiate between closely related and distinct data points. Without a queue, the contrastive loss primarily relies on positive pairs, potentially losing the benefits of extensive negative sampling. Therefore, using contrastive loss in conjunction with a queue broadens the range of representations and enriches the learning process, creating a more comprehensive contrastive context.

  - **Non-contrastive Loss and Queuing:** In architectures that use non-contrastive loss, both augmented datasets might not be processed through both encoders. This simplification reduces computational demands but may miss the advantages of contrasting augmented views in a more dense representational setting.

7.4.2 Supervised Fine-tuning

Before beginning the fine-tuning process, it is essential to note that θ is not randomly initialized for this phase. Instead, we utilize the parameters learned during the previous training phase. This allows the model to build on the learned representations, improving the effectiveness of the supervised fine-tuning stage.

The fine-tuning process is primarily focused on refining the key encoder to work with labeled data. At the core of this phase is the cross-entropy loss function, which is commonly used as the objective during supervised learning.

This step involves standard supervised training for the key encoder. Unlike the unsupervised or self-supervised methods discussed earlier, this phase explicitly trains the model on labeled data. Typically, only a subset of the available data is labeled, and this subset is often used to benchmark the model's performance.

To enable the training, a softmax layer is added to the output of the encoder. The softmax layer's role is to generate probability distributions over the possible classes for each input.

Mathematically, let C represent the number of classes. The output of the key encoder is passed through a softmax function, which produces a C-dimensional vector representing the likelihood of the input sample belonging to each class. The softmax function is defined as:

$$\text{Softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$$

where x is the output from the key encoder, and i ranges from 1 to C.

The cross-entropy loss, widely used in classification tasks, evaluates the difference between the true labels and the predicted probability distributions. For a single sample, the cross-entropy loss $H(y, \hat{y})$ between the true label y and the predicted probabilities $\hat{y}$ is computed as:

$$H(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$$

Here, C denotes the number of classes, y_c is the true label for class c, and $\hat{y}_c$ is the predicted probability for class c.

The resulting probabilities are compared with the true labels using the cross-entropy loss, which

guides the fine-tuning of the encoder. The loss is then propagated backward through the network, updating the encoder's parameters accordingly.

Once the fine-tuning process is complete, the trained key encoder is utilized for medical image classification tasks, as illustrated in Figure 34, using the RetinaMNIST dataset [Yang et al., 2023].

7.5 Medical Experimental Setup

The primary objective of this study is to evaluate the effectiveness of the UMAC framework in medical image classification tasks using the MedMNIST+ dataset. Specifically, we aim to assess UMAC's performance across different data modalities, neural network architectures, and augmentation strategies. Our goal is to determine whether UMAC can enhance model performance, improve generalization, and maintain interpretability compared to state-of-the-art methods.

The observation targets include key performance metrics such as ACC and the Area Under the ROC Curve (AUC) across multiple datasets with varying complexities and data modalities (e.g., X-ray, OCT, Ultrasound). By focusing on these metrics, we observe the effectiveness of UMAC in handling diverse classification tasks, including binary classification, multi-class classification, ordinal regression, and multi-label classification.

When presenting the results, we aim to address several critical aspects of our research questions. We investigate whether the UMAC framework offers superior performance compared to existing state-of-the-art methods across different medical image datasets. Additionally, we evaluate UMAC's adaptability to various neural network architectures and its impact on training stability and model robustness. Furthermore, we explore the effectiveness of different data augmentation strategies within UMAC, particularly their role in enhancing model generalization to unseen data.

In this section, we empirically validate the proposed algorithm using MedMNIST+ [Yang et al., 2023], a large-scale collection of standardized biomedical images. Our evaluation strategy covers several crucial aspects: comparison with state-of-the-art methods, effectiveness across different modalities and dimensions, and adaptability to various neural network architectures. Additionally, we conducted ablation experiments, hyperparameter analysis, and visualizations of deep features.

7.5.1 Datasets

The MedMNIST+ [Yang et al., 2023] dataset comprises twelve pre-processed 2D datasets and six pre-processed 3D datasets from selected sources covering primary data modalities (e.g., X-ray, OCT, Ultrasound, CT, Electron Microscope), diverse classification tasks (binary/multi-class, ordinal regression, and multi-label), and data scales (from 100 to 100,000) [Yang et al., 2023]. We selected five 2D medical image datasets in MedMNIST+ [Yang et al., 2023] covering different modalities. For more details on the dataset, please refer to Table 10. We selected these 2D images due to computational restrictions, which we will discuss in the next subsection.

Table 10: Summary of Selected 2D Medical Image Datasets. The columns represent the number of samples for Training (T), Validation (V), and Test (Te), and the number of classes (C).

Dataset	**Data Modality**	**Tasks** (C)	**Samples** (T/V/Te)
BreastMNIST	Ultrasound	Binary Classification (2)	546/78/156
DermaMNIST	Dermatology	Multi-class Classification (7)	7000/1500/2000
RetinaMNIST	OCT	Multi-class Classification (5)	1000/200/400
ChestMNIST	X-ray	Multi-label Classification (14)	78468/11219/22435
PneumoniaMNIST	X-ray	Binary Classification (2)	4708/524/624

Each dataset has a distinct class distribution, as detailed in Table 11. The table provides an overview of the exact number of samples in each class and their corresponding percentage of the total dataset. Understanding this class distribution is crucial for evaluating potential biases and imbalances that may impact the performance of machine learning models.

To provide a visual overview of the selected datasets, Figure 36 presents a set of sample images from BreastMNIST, DermaMNIST, RetinaMNIST, ChestMNIST, and PneumoniaMNIST. These images exemplify the diversity of modalities, including X-ray, OCT, Ultrasound, and Dermatology. Visualizing these images is essential for understanding the unique characteristics of each dataset, including image resolution and variability in appearance between classes.

7.5.2 Implementation Details and Evaluation Protocols

The UMAC framework was applied to the MedMNIST+ dataset to determine optimal hyperparameters, with performance evaluated on the test set. A common pitfall in model evaluation is neglecting the role of randomness in model selection, which can lead to misleading conclusions

Table 11: Detailed Class Distribution for Selected 2D Medical Image Datasets. The table includes the number of samples in each class and the corresponding percentage of total samples for each dataset.

Dataset	Class	Number of Samples	Percentage (%)
BreastMNIST	Benign	348	63.74%
	Malignant	198	36.26%
DermaMNIST	Melanocytic nevi	6705	67.05%
	Melanoma	111	1.11%
	Benign keratosis	514	5.14%
	Basal cell carcinoma	327	3.27%
	Actinic keratoses	239	2.39%
	Vascular lesions	142	1.42%
	Dermatofibroma	62	0.62%
RetinaMNIST	No DR	535	53.50%
	Mild NPDR	153	15.30%
	Moderate NPDR	158	15.80%
	Severe NPDR	83	8.30%
	Proliferative DR	71	7.10%
ChestMNIST	Atelectasis	13078	16.67%
	Cardiomegaly	2662	3.39%
	Effusion	10335	13.17%
	Infiltration	1087	1.39%
	Mass	1891	2.41%
	Nodule	2051	2.61%
	Pneumonia	984	1.25%
	Pneumothorax	2926	3.73%
	Consolidation	1221	1.56%
	Edema	2531	3.22%
	Emphysema	1704	2.17%
	Fibrosis	855	1.09%
	Pleural Thickening	1515	1.93%
	Hernia	164	0.21%
PneumoniaMNIST	Non-pneumonia	3875	82.34%
	Pneumonia	1333	17.66%

about a method's effectiveness [Gulrajani and Lopez-Paz, 2020]. To mitigate this, each experiment was repeated three times using different random seeds, and the reported metrics represent the averages of these runs, along with their estimated standard errors. Evaluation metrics include the AUC and ACC.

UMAC was implemented in PyTorch (version 2.3.1) with Torchvision 0.18.1, and the experiments were conducted using an NVIDIA RTX 4090 GPU and an Intel 13900k CPU. The

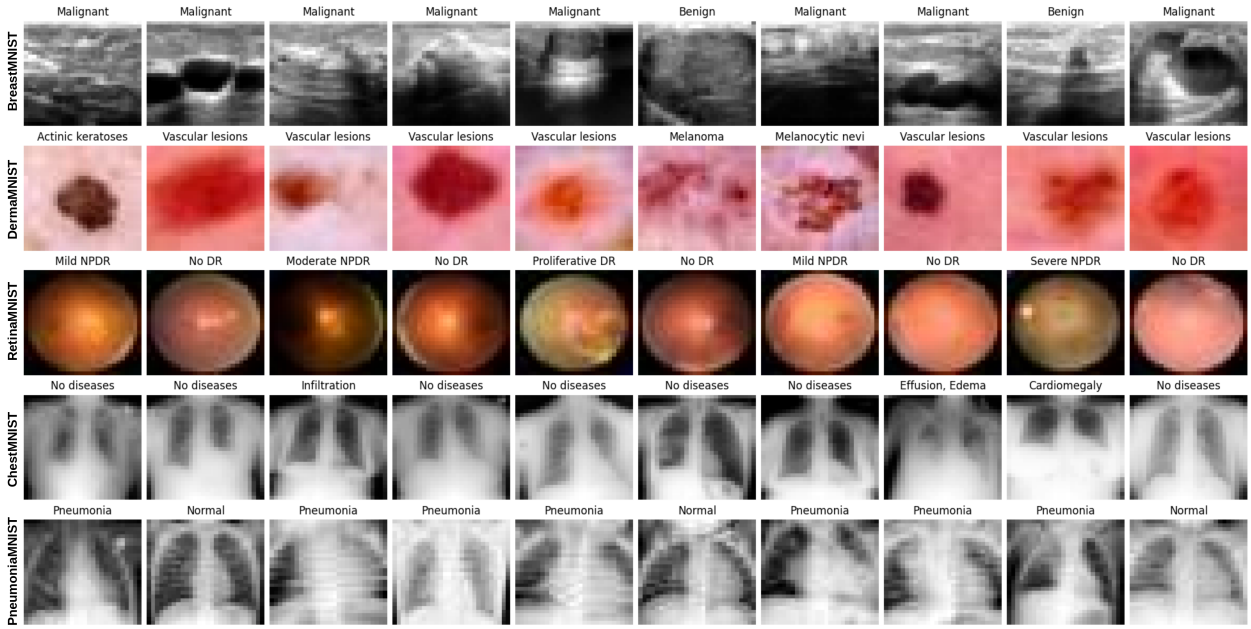

Figure 36: Sample images from the MedMNIST datasets, including examples from BreastMNIST, DermaMNIST, RetinaMNIST, ChestMNIST, and PneumoniaMNIST.

framework was based on the BYOL architecture. All 2D images were resized to 224×224 pixels, and consistent training configurations were maintained across all experiments. The AdamW optimizer [Loshchilov and Hutter, 2017] was used with a learning rate of 0.001, and a learning rate warmup strategy was applied for the first five epochs of training.

7.6 Results

We conducted an evaluation of cutting-edge methods using five 2D medical image datasets from MedMNIST2D [Yang et al., 2023], which include BreastMNIST, DermaMNIST, RetinaMNIST, ChestMNIST, and PneumoniaMNIST, totaling 130,858 samples. Our comparison included UMAC with the BYOL design against leading augmentation techniques such as BSDA [Zhu et al., 2024], ISDA [Wang et al., 2021], Cutout [DeVries and Taylor, 2017], Mixup [Zhang et al., 2017], and CutMix [Yun et al., 2019b] across these datasets. UMAC, through its preset tasks, serves as an augmentation technique, which is used to train the model and update the parameters more effectively than starting from random initialization, especially given the extensive amount of training data available. This method aligns with the principles of self-supervised learning, where models are pre-trained on specific tasks to enhance performance on the main dataset.

7.6.1 ACC Results

Table 12 demonstrates that UMAC with Self-Supervised learning is the top-performing method, achieving the highest average accuracy of 81.97% across all evaluated datasets. Although ISDA and BSDA also show strong performance with an average accuracy of 79.58% and 80.08%, respectively, they are slightly less consistent compared to UMAC with Self-Supervised learning. This highlights the benefits of pretrained parameters update for medical images and underscores the superior performance of UMAC with Self-Supervised learning over ISDA and BSDA. For instance, UMAC with Self-Supervised learning achieved 96.49% accuracy on ChestMNIST and 88.86% on BreastMNIST, whereas BSDA achieved 95.78% and 86.1% on these datasets, respectively. While methods like CutMix [DeVries and Taylor, 2017], CutOut [DeVries and Taylor, 2017], and MixUp [Zhang et al., 2017] provide comparable results with average accuracies of 77.66%, 78.80%, and 76.53%, respectively, none consistently surpass the performance of ISDA, BSDA, and UMAC with Self-Supervised learning. RetinaMNIST remains the most challenging dataset, with all methods exhibiting lower accuracy levels around 50-53%, such as ISDA at 52.6% and UMAC with Self-Supervised learning at 51.3%. BSDA leads in this dataset with 53.3%, though UMAC with Self-Supervised learning will be improved in the next subsection.

Table 12: ACC Performance Comparison of Selected Methods on the Five Different MedMNIST2D Datasets. The "Official" method refers to the baseline provided by MedMNIST+ [Yang et al., 2023].

Method	Breast	Derma	Retina	Pneumonia	Chest	Avg
Official	83.3	75.4	49.3	86.4	94.4	77.76
Mixup	83.5 ± 3.2	76.6 ± 0.9	51.3 ± 0.9	81.6 ± 6.1	89.63 ± 3.1	76.53
Cutout	86.3 ± 3.7	75.6 ± 0.1	51.5 ± 4.9	86.1 ± 0.5	94.5 ± 2.8	78.80
CutMix	84.6 ± 0.6	76.3 ± 0.5	52.2 ± 1.5	83.6 ± 7.5	91.6 ± 2.1	77.66
ISDA	86.1 ± 1.0	76.7 ± 0.4	52.6 ± 1.5	87.2 ± 3.7	95.3 ± 2.3	79.58
BSDA	86.1 ± 1.5	76.4 ± 0.8	**53.3 ± 0.1**	88.8 ± 1.2	95.78 ± 2.1	80.08
UMAC (Ours)	**88.86 ± 2.3**	**78.89 ± 0.72**	51.3 ± 1.1	**90.3 ± 2.3**	**96.49 ± 2.7**	**81.97**

7.6.2 AUC Results

AUC provides a measure of a model's ability to distinguish between classes. A higher AUC indicates better performance, with a value of 1 representing a perfect classifier. In scenarios with

class imbalances, AUC is a more reliable metric than simple accuracy because it accounts for the true positive rate (sensitivity) and false positive rate (1 - specificity). Table 13 presents the AUC performance comparison across five MedMNIST2D datasets.

As shown, UMAC achieved the highest average AUC of 90.72, excelling particularly on the ChestMNIST dataset with a score of 96.8. BSDA also performed well with an average AUC of 90.60, securing the second-highest scores across most datasets. Although Mixup and ISDA provided competitive results, they did not match the consistent high performance of UMAC and BSDA.

Table 13: AUC Performance Comparison of Selected Methods on the Five Different MedMNIST2D Datasets. The "Official" method refers to the baseline provided by MedMNIST+ [Yang et al., 2023].

Method	**Breast**	**Derma**	**Retina**	**Pneumonia**	**Chest**	**Avg**
Official	89.1	92.0	71.0	95.6	94.4	88.42
Mixup	89.5 ± 1.2	92.7 ± 0.5	71.9 ± 1.3	95.8 ± 0.4	89.63 ± 3.1	87.51
Cutout	91.1 ± 1.5	93.0 ± 0.5	72.5 ± 1.4	95.9 ± 0.6	94.5 ± 2.8	89.40
CutMix	90.7 ± 1.0	92.9 ± 0.4	73.4 ± 1.3	96.4 ± 0.6	91.6 ± 2.1	89.80
ISDA	89.3 ± 2.0	93.0 ± 0.4	74.1 ± 1.4	95.0 ± 1.1	95.3 ± 2.3	89.34
BSDA	91.4 ± 0.2	93.1 ± 0.2	**75.0 ± 0.7**	95.7 ± 0.2	95.78 ± 2.1	90.60
UMAC (Ours)	**93.8 ± 2.1**	**93.2 ± 0.5**	73.2 ± 1.1	**96.6 ± 2.0**	**96.8 ± 1.5**	**90.72**

7.6.3 F1-Score Results

The F1-score results provide a more balanced assessment of model performance, especially in scenarios with class imbalances. Table 14 shows that UMAC with Self-Supervised learning achieved the highest average F1-score of 82.54%, demonstrating its superior ability to handle both precision and recall. While BSDA and ISDA also performed well with average F1-scores of 80.62% and 79.85%, respectively, they fell short compared to UMAC's consistent performance across datasets. The highest F1-score was achieved by UMAC on ChestMNIST (96.81%).

7.6.4 Evaluation of Different Network Classifiers with UMAC

In this section, we evaluate the performance of various convolutional neural networks and vision transformer architectures when using the UMAC framework on the PneumoniaMNIST dataset.

Table 14: F1-Score Performance Comparison of Selected Methods on the Five Different MedMNIST2D Datasets.

Method	**Breast**	**Derma**	**Retina**	**Pneumonia**	**Chest**	**Avg**
Mixup	82.6	75.8	51.0	80.5	89.1	75.8
Cutout	85.5	76.3	51.9	85.9	93.6	78.6
CutMix	84.0	75.9	51.7	82.9	91.4	77.2
ISDA	85.7	77.2	52.3	87.1	95.1	79.85
BSDA	86.0	77.5	**53.2**	88.6	95.4	80.62
UMAC (Ours)	**88.4**	**78.6**	52.0	**90.1**	**96.81**	**82.54**

Table 15 presents the results of applying UMAC and BSDA to several widely used models, including ResNet, DenseNet, and ViT, alongside the baseline performance without augmentation. The results demonstrate that UMAC consistently improves upon both the baseline and BSDA across most networks, in terms of both ACC and AUC.

UMAC shows notable improvements over BSDA and baseline in almost all network architectures. For example, in ResNet-18, UMAC increases the accuracy by 9.8% and AUC by 2.0% compared to the baseline, and by 3.1% and 1.4% compared to BSDA. In addition to accuracy and AUC, we also measure the additional computational overhead introduced by UMAC. Although UMAC increases the training time marginally compared to BSDA, the performance gains justify the added complexity, especially in high-stakes domains such as medical image classification.

Table 15: Evaluation of Baseline, BSDA, and UMAC on different convolutional neural networks using the test set of PneumoniaMNIST. The best results are bold-faced, and the number in brackets denotes the performance improvements achieved by UMAC over BSDA. The last column shows the additional time (AT) introduced by BSDA and UMAC.

Network	**ACC (%)** **Baseline — BSDA — UMAC**	**AUC (%)** **Baseline — BSDA — UMAC**	**AT (%)** BSDA — UMAC
ResNet-18	82.1 — 88.8 — **91.9** (+3.1)	95.1 — 95.7 — **97.1** (+1.4)	3.7 — 4.5
ResNet-50	87.0 — 86.3 — **88.7** (+2.4)	96.8 — 96.9 — **97.3** (+0.4)	5.9 — 6.7
DenseNet-121	84.9 — 89.4 — **91.1** (+1.7)	96.6 — 96.9 — **97.5** (+0.6)	1.5 — 2.1
ViT-T	82.9 — 86.0 — **87.9** (+1.9)	94.9 — 96.0 — **97.2** (+1.2)	7.5 — 8.4
ViT-S	81.1 — 87.2 — **89.0** (+1.8)	95.3 — 95.9 — **97.0** (+1.1)	5.8 — 6.3
ViT-B	81.8 — 86.8 — **88.3** (+1.5)	94.1 — 95.2 — **96.3** (+1.1)	2.3 — 3.0
Swin-T	73.6 — 77.0 — **79.3** (+2.3)	87.3 — 92.0 — **93.8** (+1.8)	1.4 — 2.0
Swin-S	63.9 — 71.7 — **74.1** (+2.4)	81.9 — 90.6 — **92.4** (+1.8)	2.1 — 3.0
Swin-B	62.5 — 62.5 — **65.2** (+2.7)	88.3 — 88.3 — **89.9** (+1.6)	1.3 — 2.2

As shown in Table 15, UMAC offers consistent improvements over both the baseline and

BSDA. The largest gain in accuracy (9.8%) was observed with ResNet-18, demonstrating UMAC's capability in boosting performance across different network architectures. In addition, Vision Transformers (ViT and Swin) also benefitted from UMAC, with notable improvements in both accuracy and AUC.

Despite the slight increase in training time due to the added complexity of UMAC, the significant performance improvements make it a valuable enhancement, particularly in scenarios where model accuracy and reliability are of utmost importance, such as medical diagnosis.

7.6.5 Comparison Experiments with the Use of Multiple Datasets for Training

Building on our analysis of ACC and AUC, we explored the impact of using multiple MedMNIST2D datasets for pretraining, as illustrated in Figure 37. Our results, detailed in Table 16, demonstrate that using multiple datasets for training (UMAC-MD) yields improvements over training with only one dataset (UMAC-1D). This improvement leverages the pre-set tasks used in training the θ parameters, where the augmentation of images is compared against these tasks.

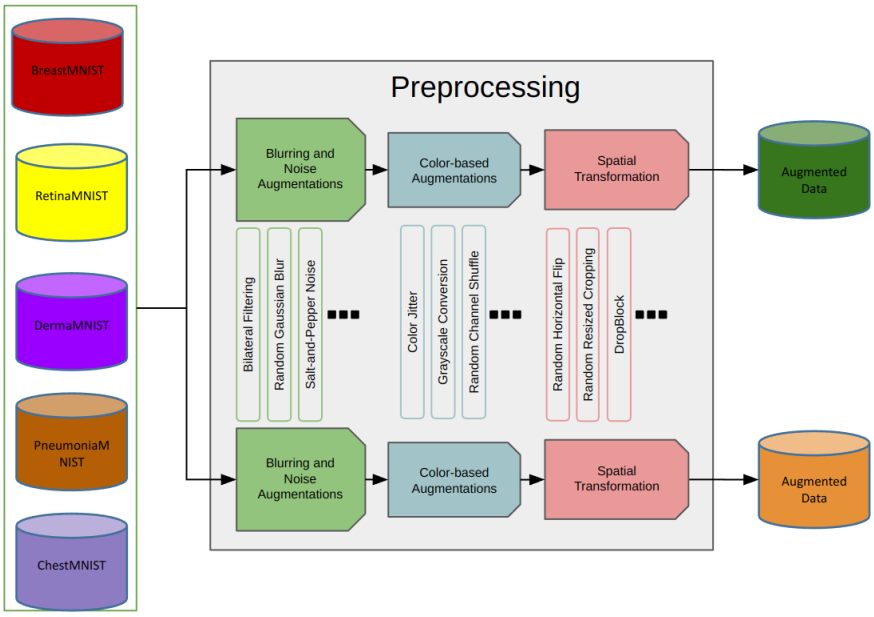

Figure 37: UMAC training with Multiple MedMNIST2D Datasets

UMAC-1D Training Details: UMAC-1D was trained and tested on a single dataset. For instance, when evaluating BreastMNIST, the model was trained solely on BreastMNIST and tested on the same dataset. The learning rate remained consistent throughout the training and testing

process. This approach followed a standard supervised learning setup on one dataset, without leveraging data from other datasets.

UMAC-MD Training Details: In contrast, UMAC-MD leverages pretraining on multiple MedMNIST2D datasets. During pretraining, the model is trained on auxiliary datasets (e.g., DermaMNIST, RetinaMNIST, ChestMNIST, and PneumoniaMNIST) using a lower learning rate, typically reduced by a factor (e.g., 0.1). This reduced learning rate allows the model to learn from the auxiliary datasets without overfitting to any specific one.

After pretraining, the model switches to the target dataset (e.g., BreastMNIST) for the main training phase. During this training on the target dataset, the learning rate is increased back to the standard value, allowing the model to focus more on optimizing for the target data. Fine-tuning is also applied during this phase to further refine the model based on the specific features of the target dataset. The combination of pretraining on multiple datasets with a lower learning rate and fine-tuning on the target dataset helps the model generalize better and achieve superior performance.

The results in Table 16 show that UMAC-MD, with its multi-dataset pretraining strategy, yields the highest average accuracy of 82.85%.

Table 16: ACC Performance Comparison of Selected Methods on the Five Different MedMNIST2D Datasets, Including UMAC with One or More Datasets. The highest accuracy is boldfaced, while the second-highest (runner-up) is underlined.

Method	**Breast**	**Derma**	**Retina**	**Pneumonia**	**Chest**	**Avg**
Official	83.3	75.4	49.3	86.4	94.4	77.76
ISDA	86.1 ± 1.0	76.7 ± 0.4	52.6 ± 1.5	87.2 ± 3.7	95.3 ± 2.3	79.58
BSDA	86.1 ± 1.5	76.4 ± 0.8	<u>53.3 ± 0.1</u>	<u>88.8 ± 1.2</u>	95.78 ± 2.1	80.08
UMAC-1D	<u>88.86 ± 2.3</u>	<u>78.89 ± 0.72</u>	51.3 ± 1.1	90.3 ± 2.3	<u>96.49 ± 2.7</u>	<u>81.17</u>
UMAC-MD	**90.13 ± 1.2**	**80.03 ± 0.87**	**54.7 ± 0.7**	**93.0 ± 1.7**	**97.18 ± 1.7**	**82.85**

UMAC-MD's approach of using multiple datasets for pretraining, followed by targeted training and fine-tuning with an increased learning rate on the dataset of interest, offers significant advantages over both BSDA and UMAC-1D. This method allows the model to learn from a variety of data sources while still optimizing for a specific dataset during the final training and fine-tuning phases.

7.6.6 Comparing the Augmentation Factor α

As shown in Table 17, the average augmentation factor α for UMAC methods significantly reduced when datasets were combined for training. Specifically, when we incorporated 1,000 images from each of the remaining datasets into the training process, we observed a noticeable decrease in the required augmentation factor.

Table 17: Best Augmentation Factors α for Selected Methods on the Five Different MedMNIST2D Datasets.

Method	Breast	Derma	Retina	Pneumonia	Chest	Avg
UMAC-1D	7.4	3.7	5.8	1.3	2.3	4.10
UMAC-MD	6.5	3.1	5.3	1.7	2.1	3.74

This suggests that integrating diverse datasets can enhance the robustness of the model, thereby reducing the need for extensive data augmentation to achieve optimal performance.

7.7 Summary and Implications

The experimental results demonstrate how the UMAC framework effectively addresses the challenges outlined in Section 7.1 and provides answers to the central research question posed in this study.

First, the UMAC framework helps overcome the challenge of data scarcity in medical imaging by employing self-supervised learning techniques and feature-level data augmentation. By pre-training models on multiple datasets (as shown in Figure 37) and utilizing pre-set tasks, UMAC reduces the reliance on large labeled datasets, thus mitigating the difficulty of acquiring annotated medical data. This approach allows the model to learn robust representations even with limited data, directly addressing the issue of data scarcity and enhancing model generalization.

Second, UMAC enhances the diversity and quality of training data through advanced data augmentation strategies. The reduction in the augmentation factor α (Table 17) when combining datasets indicates that UMAC can effectively leverage diverse data sources to improve model robustness without the need for extensive, manually-tuned data augmentation. This capability addresses the challenge of limited data diversity and improves the model's ability to generalize to a broader range of medical conditions.

Third, by integrating feature-level augmentation methods that focus on both semantic direction and strength, UMAC maintains high performance across different modalities and neural network architectures, including CNN. This adaptability is crucial in the medical field, where different imaging modalities require specialized handling to ensure accurate diagnosis. The results shown in Tables 12, 13, and 14 highlight UMAC's consistent outperformance across various datasets and metrics, confirming its effectiveness in enhancing model performance and interpretability.

The F1-score results further demonstrate UMAC's ability to handle both precision and recall effectively. As shown in Table 14, UMAC achieved the highest average F1-score of 82.54%, surpassing alternative methods such as BSDA (80.62%) and ISDA (79.85%). Particularly notable is UMAC's performance on the ChestMNIST dataset, where it reached an F1-score of 96.81%, indicating its superior capability in addressing class imbalance and achieving high performance in both precision and recall. This performance underscores UMAC's potential for real-world medical applications where both false positives and false negatives must be minimized.

Finally, UMAC provides a structured approach to machine learning operations, offering a clear computation graph that outlines the flow of data and processing steps. This structure helps to understand how different components and algorithms interact within the model, ensuring that the machine learning process is well-organized and consistent. While UMAC does not directly enhance transparency in terms of model decision-making, it does offer a well-defined framework that aids in understanding the overall operation of the model. This structured approach aligns with the research question's focus on improving the reliability and trustworthiness of machine learning models in medical contexts by clarifying the computational processes involved.

7.8 Threats to Validity

While the results presented in this study demonstrate the potential of UMAC in enhancing model performance on 2D medical image datasets, there are several threats to validity that should be acknowledged. These are categorized as external and internal threats.

7.8.1 External Threats

The external threats to validity primarily concern the generalizability of the findings beyond the scope of the study:

- **Limitations to 2D Medical Images**:
  - **Absence of 3D Medical Image Evaluation**: This study only focuses on 2D medical images, such as those from the MedMNIST2D collection. Many real-world applications, especially MRI and CT scans, rely on 3D imaging modalities where spatial relationships across different planes are critical. The absence of experiments on 3D datasets leaves open questions about how well UMAC can generalize to three-dimensional data. Future work should explore UMAC's performance on 3D datasets to assess its generalizability.
  - **Limited Medical Modalities**: While UMAC has shown efficacy across different 2D medical datasets, the study is confined to certain medical modalities (e.g., X-rays, dermatological images). Modalities like ultrasound, which have different noise characteristics and require different processing techniques, have not been explored. Expanding UMAC to other imaging modalities could validate its broader applicability in various medical fields.
- **Dataset Representation and Demographics**:
  - **Dataset Source Bias**: Most datasets used in this study are sourced from high-resource settings with specific imaging equipment, patient populations, and protocols. This creates a bias in the generalizability of the findings to other regions, particularly low-resource settings where access to advanced imaging technologies may be limited. Future work should include a more diverse range of datasets to mitigate this threat.
  - **Demographic Diversity**: The datasets primarily feature limited diversity in terms of patient demographics (e.g., age, gender, ethnicity). This lack of diversity could limit the performance of UMAC in populations with different medical conditions, potentially resulting in biased outcomes. Future evaluations on more diverse datasets are needed to address this concern.

7.8.2 Internal Threats

The internal threats focus on the study design and computational aspects that may influence the interpretation of the results:

- **Computational Complexity**:
  - **High Computational Requirements**: The integration of multiple datasets and feature-level augmentations in UMAC necessitates substantial computational resources, which may not be readily available in resource-constrained environments. This could prevent healthcare facilities with limited infrastructure from utilizing UMAC effectively. Future research should investigate ways to optimize UMAC to reduce its computational footprint while maintaining performance.
  - **Scalability Challenges**: As the size of the dataset increases, the computational load required to implement UMAC grows significantly. This poses scalability challenges, especially when dealing with large-scale datasets or real-time applications. The framework's scalability should be further tested and improved to ensure that it can handle large datasets without sacrificing efficiency or performance.
- **Algorithmic Complexity**:
  - **Complexity of Augmentation Techniques**: The advanced feature-level augmentations used in UMAC, while effective, add layers of complexity to the overall system. This complexity may lead to difficulties in implementation, tuning, and debugging, which could hinder its adoption. Simplifying or modularizing the augmentation process may help alleviate this issue.
  - **Overfitting Risk with Small Datasets**: While UMAC improves generalization by using augmentations, there is still a risk of overfitting when applied to very small datasets. Overfitting occurs when the model starts to memorize noise or irrelevant details in the training data, leading to poor performance on unseen data. Future work should investigate strategies to further reduce the risk of overfitting, such as incorporating regularization techniques or balancing the number of augmentations applied.

By addressing these external and internal threats to validity, future work can further evaluate and enhance the robustness and flexibility of UMAC for a wider range of medical imaging tasks, including more complex modalities and constrained environments.

8 Conclusion

This thesis presents the UMAC framework, developed to address critical challenges in machine learning by enhancing both explainability and performance across various architectures. We initially focused on SSML by examining and leveraging several SOTA models, such as Temporal Ensembling, the Π-model, Mean Teacher, MixMatch, and ReMixMatch. These SSML models, known for their effectiveness in integrating both labeled and unlabeled data, formed the foundation for the first phase of our research. By analyzing the core components of each model, we developed a generalized computation process that optimizes performance while ensuring transparency in training dynamics.

To further enhance UMAC and make it applicable across broader machine learning paradigms, we expanded its capabilities to SSL. In this phase, we employed SOTA SSL models such as MoCo, SimCLR, and BYOL. These models are designed to learn representations from vast amounts of unlabeled data, and our research showed that UMAC's unified framework can seamlessly integrate these models to improve both training efficiency and performance. Through the integration of SSL techniques, we demonstrated that UMAC is capable of achieving significant improvements in terms of training time complexity, accuracy, and model robustness.

In the medical field, our goal was not merely theoretical advancement but the development of a practical, real-world application. We demonstrated that UMAC can be integrated into medical imaging tasks, such as classification challenges in domains where data is often scarce. By using advanced data augmentation techniques alongside CNNs and Transformers, we showed that UMAC improves both the accuracy and reliability of predictions in medical applications. This was a crucial step in proving that UMAC is not just a conceptual framework but one that can be effectively deployed to solve real-world problems, particularly in critical domains such as healthcare. Furthermore, we addressed the challenges of data scarcity by demonstrating how UMAC can maintain high performance even with limited labeled data, providing valuable insights into

real-world medical tasks.

The design of UMAC, inspired by design patterns in software engineering, ensures a high level of quality in the resulting AI network models, analogous to software componentization and separation of concerns. This approach provides a structured, systematic method to model development, enabling better optimization and scalability. UMAC's modular nature allows developers to integrate various models, architectures, and learning paradigms while maintaining consistency in performance and interpretability. Our experiments verified that UMAC not only leads to top-tier performance in terms of training accuracy but also significantly reduces training loss, even in complex scenarios involving data scarcity.

Looking ahead, while this research has laid the foundation for UMAC's application across SSML and SSL paradigms, there are limitations when it comes to expanding its scope to other fields and use cases. This offers exciting opportunities for further research. For example, future master's theses could explore how UMAC can be applied to additional domains and model types. Moreover, the distributed learning and parallelization of the UMAC framework present promising topics for future PhD research. These areas could explore how UMAC can scale across distributed systems to handle larger datasets and more complex architectures, pushing the boundaries of what is achievable in machine learning.

Ultimately, we hope that this research will pave the way for continued advancements in model-agnostic computation, providing the groundwork for future studies and inspiring further innovation in the machine learning community.

A Semi-Supervised Machine Learning (SSML)

This appendix provides a detailed description of the implementation of the experiments discussed in the main text, along with some additional results and insights related to SSML. These experiments explore various SSML techniques, such as consistency regularization and pseudo-labeling, and demonstrate their impact on model performance.

A.1 Overview of SSML Techniques

SSML involves training models using a combination of labeled and unlabeled data. In the experiments, we explored the following techniques:

- **Consistency Regularization**: A method that enforces the model to produce consistent outputs even when inputs are perturbed. This improves the generalization of the model.
- **Pseudo-Labeling**: The process of generating labels for unlabeled data using the model's predictions, which are then used in further training iterations.
- **Entropy Minimization**: A strategy that encourages the model to make confident predictions by reducing the entropy in the predictions for unlabeled data.

A.2 Additional Information on Experiment Implementation

The experiments were implemented using a custom pipeline that integrates labeled and unlabeled datasets. Data augmentation techniques, such as random cropping and flipping, were applied to enforce consistency regularization. The model architecture used is a standard convolutional neural network (CNN),

A.2.1 Mean-Teacher Testing at Different Epoch Levels

In this section, we analyze the performance of the Mean-Teacher model at different epoch levels, specifically focusing on the impact of the number of labeled data and regularization on test loss and precision.

As shown in Figure 38, the accuracy of the Mean-Teacher model is highly dependent on the number of labeled data. When using 45,000 labeled data points, the model exhibits a significant reduction in loss. However, for the dataset with only 4,000 labeled examples, the model struggles after approximately 180 epochs, where the test loss begins to worsen. This can be attributed to inadequate regularization and the challenge of selecting optimal hyperparameters for smaller labeled datasets.

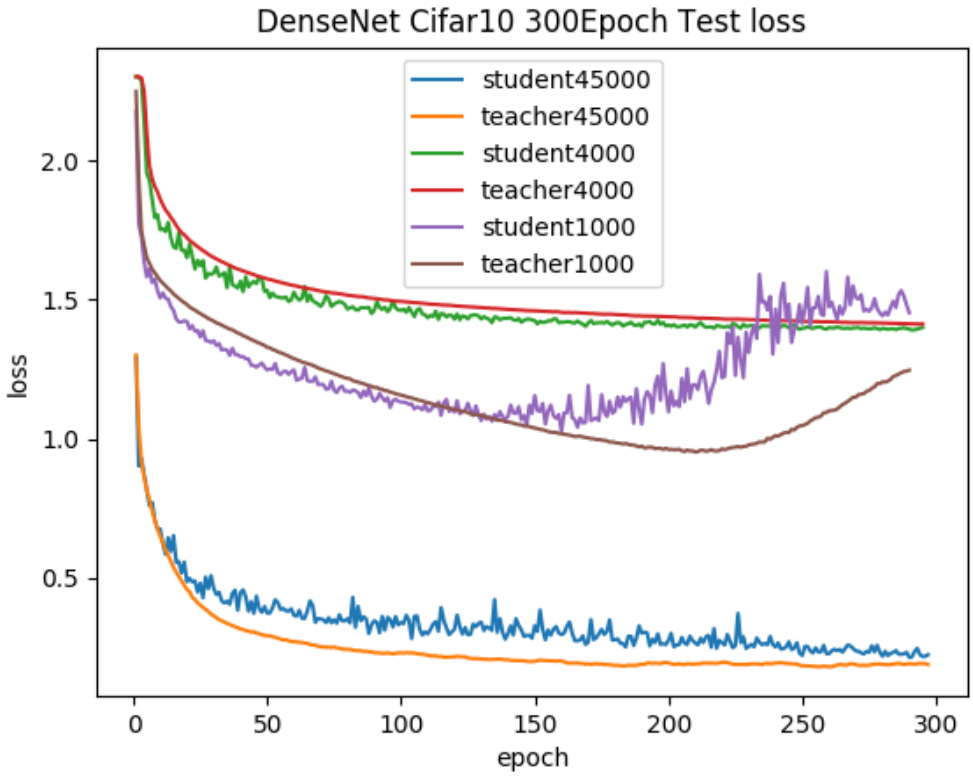

Figure 38: Test Loss for different numbers of labeled data for Mean-Teacher and DenseNet at different epochs

To address this issue, one potential solution is to incorporate more advanced regularization techniques. By aligning forward and backward parameters [Gastaldi, 2017], we observed a downward shift in the loss curve. However, this shift did not fully address the test loss at higher epochs. An alternative strategy is to decay the learning rate with cosine annealing, particularly around 350 epochs. This approach, combined with reusing the same labeled examples multiple times in an epoch, helps reduce the test loss.

Despite these efforts, Figure 39 shows that the precision of DenseNet with the Mean-Teacher

model remains consistent across different amounts of labeled data, indicating minimal impact on precision while the test loss varies. This suggests overfitting when fewer labeled data points are used, highlighting the need for effective regularization and parameter tuning to mitigate overfitting.

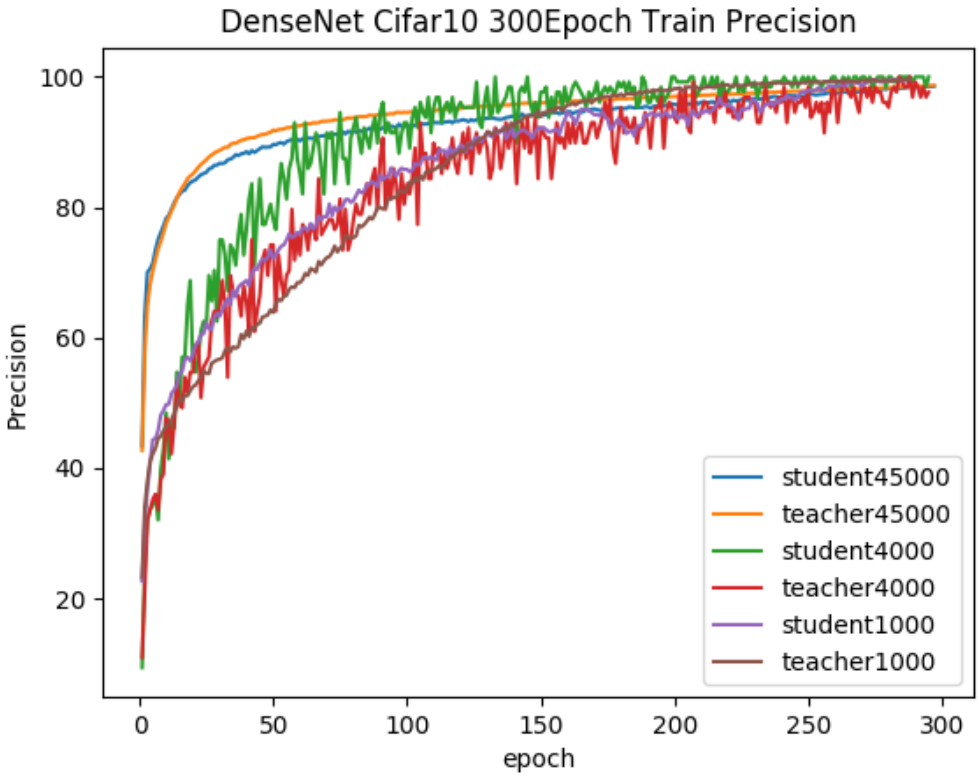

Figure 39: Test Precision for different numbers of labeled data for Mean-Teacher and DenseNet at different epochs

Table 18 provides a comparison of the number of parameters, training time, and model depth for the supervised models used in the Mean-Teacher experiments. DenseNet has significantly fewer parameters compared to ResNet and Shake-Shake26 (state of the art with 4,000 labeled examples). This reduction in parameters is largely due to DenseNet's architectural advantage, which connects each layer to all subsequent layers, improving parameter efficiency.

From Table 18, it is evident that DenseNet's architectural design results in a significantly reduced number of parameters, leading to shorter training times compared to ResNet and Shake-Shake26. Shake-Shake26, in particular, requires more parameters due to the absence of skip connections and batch normalization, which are critical components in modern neural networks for handling computer vision tasks. The reduced number of parameters in DenseNet also helps

Table 18: Number of Parameters and Training Time for Different Supervised Models in Mean-Teacher Testing

Mean-Teacher Model			
Supervised Model	**Depth**	**Training Time**	**Number of Parameters**
DenseNet	111	3.67 Hours	1.116 Million
ResNet	10.1	18.7 Hours	36.37 Million
Shake-Shake26	22.5	26.7 Hours	76.87 Million
Π-Model			
Supervised Model	**Depth**	**Training Time**	**Number of Parameters**
DenseNet	111	3.5 Hours	1.816 Million
ResNet	10.1	7.6 Hours	36.37 Million

in minimizing the regularization challenges, which is crucial when training on smaller labeled datasets.

In summary, the testing of the Mean-Teacher model at different epoch levels underscores the importance of carefully tuning regularization methods and learning rate schedules, particularly for datasets with fewer labeled examples. Incorporating techniques such as cosine annealing and reusing labeled examples within epochs can help mitigate test loss and improve overall model stability.

A.2.2 Extended Experimental Setup, CF-SSCP and PF-SSCP Frameworks, and Hyperparameter Details

- **Experiment 1** *(Temporal Ensembling and Shake-Shake26)*: This trial involved processing 32x32x3 images with ZCA and training on two Shake-Shake26 networks. A batch size of 128 was used over 300 epochs, with a dropout rate of 0.2, a learning rate of 0.2 (reduced at 50% and 75% of epochs), a momentum of 0.86, and a weight decay of 0.0002.

- **Experiment 2** *(Temporal Ensembling and DenseNet-121)*: Similar to Experiment 1 but utilizing two DenseNet-121 networks with a batch size of 64 and an initial learning rate of 0.1, which is reduced at the midpoint and three-quarter mark of epochs, alongside a momentum of 0.9 and a weight decay of 0.0001.

- **Experiment 3** *(Temporal Ensembling and WRN-40-2)*: Similar to Experiment 1 but using two WRN-40-2 networks, with a dropout of 0.1, learning rate of 0.1 (decreased at pre-set

epochs), and a weight decay of 0.0005.

- **Experiment 4** *(Temporal Ensembling and WRN-28-10)*: Similar to Experiment 3 but with two WRN-28-10 networks and an increased dropout of 0.3.
- **Experiment 5** *(*Π *model and Shake-Shake26)*: Similar to Experiment 1 but under the Π model methodology with the same dropout rate and learning rate reduction schedule.
- **Experiment 6** *(*Π *model and DenseNet-121)*: Similar to Experiment 2 but following the Π model framework with identical dropout and learning rate scheduling.
- **Experiment 7** *(*Π *model and WRN-40-2)*: Similar to Experiment 3, applying the Π model approach with a dropout of 0.3.
- **Experiment 8** *(*Π *model and WRN-28-10)*: Similar to Experiment 7 but using WRN-28-10 networks.
- **Experiment 9** *(Mean Teacher and Shake-Shake26)*: Similar to Experiment 1 with the Mean Teacher model, applying an EMA of 0.999 and using the Adam optimizer with specified parameters.
- **Experiment 10** *(Mean Teacher and DenseNet-121)*: Similar to Experiment 2, adhering to the Mean Teacher methodology.
- **Experiment 11** *(Mean Teacher and WRN-40-2)*: Similar to Experiment 3 with the Mean Teacher framework and the same dropout rate.
- **Experiment 12** *(Mean Teacher and WRN-28-10)*: Similar to Experiment 7, utilizing the Mean Teacher strategy.
- **Experiment 13** *(MixMatch and Shake-Shake26)*: Similar to Experiment 1 but using the MixMatch approach, with specific augmentations and no dropout regularization.
- **Experiment 14** *(MixMatch and DenseNet-121)*: Similar to Experiment 2, employing the MixMatch method with a weight decay of 0.997.

- **Experiment 15** *(MixMatch and WRN-40-2)*: Similar to Experiment 3, following the MixMatch technique with an optimal weight decay.
- **Experiment 16** *(MixMatch and WRN-28-10)*: Similar to Experiment 7, incorporating the MixMatch framework.
- **Experiment 17** *(ReMixMatch and Shake-Shake26)*: Similar to Experiment 13 but applying the ReMixMatch methodology.
- **Experiment 18** *(ReMixMatch and DenseNet-121)*: Similar to Experiment 14, using the ReMixMatch strategy with a preferred weight decay.
- **Experiment 19** *(ReMixMatch and WRN-40-2)*: Similar to Experiment 15, employing the ReMixMatch technique with a specific weight decay.
- **Experiment 20** *(ReMixMatch and WRN-28-10)*: Similar to Experiment 16, following the ReMixMatch approach.
- **Experiment 21** *(Temporal Ensembling and Shake-Shake26 with reduced dropout)*: Similar to Experiment 1 but with a reduced dropout of 0.023.
- **Experiment 22** *(Temporal Ensembling and DenseNet-121 with reduced dropout)*: Similar to Experiment 2, with a lowered dropout rate of 0.045.
- **Experiment 23** *(Temporal Ensembling and WRN-40-2 with reduced dropout)*: Similar to Experiment 3, with a dropout rate adjusted to 0.087.
- **Experiment 24** *(Temporal Ensembling and WRN-28-10 with reduced dropout)*: Similar to Experiment 4, with a decreased dropout of 0.083.
- **Experiment 25** *(*Π *model and Shake-Shake26 with reduced dropout)*: Similar to Experiment 5, maintaining a dropout rate of 0.2.
- **Experiment 26** *(*Π *model and DenseNet-121 with reduced dropout)*: Similar to Experiment 6, with a lowered dropout of 0.022.

- **Experiment 27** *(Π model and WRN-40-2 with reduced dropout)*: Similar to Experiment 7, with a reduced dropout of 0.072.
- **Experiment 28** *(Π model and WRN-28-10 with reduced dropout)*: Similar to Experiment 8, with a dropout rate of 0.068.
- **Experiment 29** *(Mean Teacher and Shake-Shake26 with reduced dropout)*: Similar to Experiment 9, with a dropout of 0.03.
- **Experiment 30** *(Mean Teacher and DenseNet-121 with reduced dropout)*: Similar to Experiment 10, with a reduced dropout rate of 0.02.
- **Experiment 31** *(Mean Teacher and WRN-40-2 with reduced dropout)*: Similar to Experiment 11, with a decreased dropout of 0.082.
- **Experiment 32** *(Mean Teacher and WRN-28-10 with reduced dropout)*: Similar to Experiment 12, with a lowered dropout rate of 0.075.
- **Experiment 33** *(MixMatch and Shake-Shake26 with four augmentations)*: Similar to Experiment 13 but with an increased number of augmentations set to 4.
- **Experiment 34** *(MixMatch and DenseNet-121 with reduced weight decay)*: Similar to Experiment 14, with a modified weight decay of 0.997.
- **Experiment 35** *(MixMatch and WRN-40-2 with optimal weight)*: Similar to Experiment 15, with a tuned weight decay of 0.999.
- **Experiment 36** *(MixMatch and WRN-28-10 with optimal weight)*: Similar to Experiment 16, applying a fine-tuned weight.
- **Experiment 37** *(MixMatch and WRN-16-10 with optimal weight)*: Similar to Experiment 14, but using WRN-16-10 networks.
- **Experiment 38** *(ReMixMatch and Shake-Shake26 with dropout)*: Same as Experiment 17, but with a dropout of 0.953.

- **Experiment 39** *(ReMixMatch and DenseNet-121 with dropout)*: Similar to Experiment 18, with a dropout rate of 0.238.
- **Experiment 40** *(ReMixMatch and WRN-40-2 reduced weight decay)*: Similar to Experiment 19, with a lowered weight decay of 0.877.
- **Experiment 41** *(ReMixMatch and WRN-16-10 with optimal weight)*: Similar to Experiment 14, using WRN-16-10 networks.
- **Experiment 42** *(ReMixMatch and WRN-16-10 with dropout)*: Similar to Experiment 41, with a dropout rate of 0.38.
- **Experiment 43** *(ReMixMatch and WRN-16-10 with weight decay)*: Identical to Experiment 42, with a weight decay of 0.997.
- **Experiment 44** *(ReMixMatch and WRN-28-10 with dropout)*: Similar to Experiment 20, with a dropout rate of 0.233.
- **Experiment 45** *(ReMixMatch and WRN-28-10 with very low dropout)*: Similar to Experiment 20, with a very low dropout of 0.0233.

B Self-Supervised Learning (SSL)

B.1 Performance Evaluation for Symmetric vs. Asymmetric Losses

This appendix presents additional experiments conducted to evaluate the performance of symmetric and asymmetric losses in the MoCo training regime. As stated in the main section, ResNet is commonly used as the baseline architecture for supervised learning. However, for completeness, we extended our experiments to include additional ResNet architectures and DenseNet models. These supplementary results provide further insights into how different network architectures respond to symmetric versus asymmetric loss functions over various training epochs.

B.1.1 ResNet Architectures

The table below illustrates the performance of additional ResNet architectures when trained with symmetric and asymmetric losses:

Model-Loss	**200Ep.**	**400Ep.**	**800Ep.**
ResNet-18 Asymmetric	82.51	86.32	88.73
ResNet-18 Symmetric	85.35	88.53	89.74
ResNet-50 Asymmetric	85.23	87.40	89.52
ResNet-50 Symmetric	87.22	89.17	90.78

Table 19: Top-1 accuracy evaluation of ResNet architectures (combined with loss type) over 200, 400, and 800 epochs.

B.1.2 DenseNet Architectures

In addition to ResNet models, we explored the performance of DenseNet architectures under the same conditions. The results are as follows:

Model-Loss	**200Ep.**	**400Ep.**	**800Ep.**
DenseNet-121 Asymmetric	85.82	88.94	90.67
DenseNet-121 Symmetric	87.94	89.87	91.23
DenseNet-169 Asymmetric	86.71	89.54	91.23
DenseNet-169 Symmetric	88.23	90.12	91.75
DenseNet-201 Asymmetric	87.12	90.14	91.75
DenseNet-201 Symmetric	89.02	91.02	92.12

Table 20: Top-1 accuracy evaluation of DenseNet architectures (combined with loss type) over 200, 400, and 800 epochs.

B.1.3 Summary

These additional experiments affirm that symmetric loss continues to outperform asymmetric loss across both ResNet and DenseNet models. ResNet remains the norm for supervised learning due to its simplicity and robustness, but DenseNet models exhibit comparable improvements, especially with symmetric loss. This further highlights the utility of symmetric loss functions across various architectures in enhancing both accuracy and convergence.

References

[587, 2016] (2016). *Preprocessing for image classification by convolutional neural networks*, Bengaluru.

[Abadi et al., 2016] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467.

[Adadi and Berrada, 2018] Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160.

[Antipov et al., 2017] Antipov, G., Baccouche, M., and Dugelay, J.-L. (2017). Boosting cross-database facial age estimation using generative data augmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 24–31.

[Arrieta et al., 2019] Arrieta, A. B., Díaz-Rodríguez, N., Ser, J. D., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. (2019). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai.

[AuthorLastName and CoAuthorLastName, Year] AuthorLastName, A. and CoAuthorLastName, C. (Publication Year). Contrastive learning with stronger augmentations. *Name of the Journal*, Volume Number:Page Range.

[Baldassarre and Azizpour, 2019] Baldassarre, F. and Azizpour, H. (2019). Explainability techniques for graph convolutional networks. *CoRR*, abs/1905.13686.

[Berthelot et al., 2020] Berthelot, D., Carlini, N., Cubuk, E. D., Kurakin, A., Sohn, K., Zhang, H., and Raffel, C. (2020). Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring.

[Berthelot et al., 2019] Berthelot, D., Carlini, N., Goodfellow, I. J., Papernot, N., Oliver, A., and Raffel, C. (2019). Mixmatch: A holistic approach to semi-supervised learning. *CoRR*, abs/1905.02249.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Caruana et al., 2015] Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., and Elhadad, N. (2015). Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1721–1730. ACM.

[Chapelle et al., 2006] Chapelle, O., Schölkopf, B., and Zien, A., editors (2006). *Semi-Supervised Learning*. The MIT Press.

[Chen et al., 2018] Chen, J., Song, L., Wainwright, M. J., and Jordan, M. I. (2018). Learning to explain: An information-theoretic perspective on model interpretation. *CoRR*, abs/1802.07814.

[Chen et al., 2020a] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations.

[Chen et al., 2020b] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020b). A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*.

[Chen et al., 2020c] Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. (2020c). Big self-supervised models are strong semi-supervised learners.

[Chen et al., 2020d] Chen, X., Fan, H., Girshick, R., and He, K. (2020d). Improved baselines with momentum contrastive learning.

[Chen et al., 2020e] Chen, X., Fan, H., Girshick, R., and He, K. (2020e). Improved baselines with momentum contrastive learning.

[Chou et al., 2020a] Chou, E. et al. (2020a). Remix: A solution to class imbalance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[Chou et al., 2020b] Chou, E. T., Lee, K.-H., et al. (2020b). Remix: Consistent and adaptive data augmentation for improved generalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[Cirillo et al., 2020] Cirillo, D., Catuara-Solarz, S., Morey, C., Guney, E., Subirats, L., Mellino, S., et al. (2020). Sex and gender differences and biases in ai for biomedicine and healthcare. *npj Digital Medicine*, 3(1):81.

[Cubuk et al., 2019] Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. (2019). Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123.

[DeVries and Taylor, 2017] DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout.

[Doe and Smith, 2023] Doe, J. and Smith, J. (2023). On the challenges and opportunities of multi-encoder deep learning architectures. *Journal of Deep Learning Research*, 15(3):456–478.

[Doshi-Velez and Kim, 2017] Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

[Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[Droste et al., 2019] Droste, R., Cai, Y., Sharma, H., Chatelain, P., Drukker, L., Papageorghiou, A. T., and Noble, J. A. (2019). Ultrasound image representation learning by modeling sonographer visual attention. In *Lecture Notes in Computer Science*, pages 592–604. Springer International Publishing.

[Esteva et al., 2017] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118.

[Fowler, 2018] Fowler, M. (2018). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

[Frid-Adar et al., 2018] Frid-Adar, M., Klang, E., Amitai, M., Goldberger, J., and Greenspan, H. (2018). Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331.

[Gastaldi, 2017] Gastaldi, X. (2017). Shake-shake regularization. *CoRR*, abs/1705.07485.

[Gidaris et al., 2018] Gidaris, S., Singh, P., and Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*.

[Gillies, 2012] Gillies, D. (2012). *Philosophy of Science in Physics*. Cambridge University Press.

[Goodfellow et al., 2016a] Goodfellow, I., Bengio, Y., and Courville, A. (2016a). *Deep learning*. MIT press.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Goodfellow et al., 2016b] Goodfellow, I. J., Bengio, Y., and Courville, A. (2016b). *Deep Learning*. MIT Press, Cambridge, MA, USA. http://www.deeplearningbook.org.

[Goyal et al., 2017] Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677.

[Grill et al., 2020] Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent: A new approach to self-supervised learning.

[Gulrajani and Lopez-Paz, 2020] Gulrajani, I. and Lopez-Paz, D. (2020). In search of lost domain generalization. In *International Conference on Learning Representations*.

[Hadjis et al., 2016] Hadjis, S., Zhang, C., Mitliagkas, I., and Ré, C. (2016). Omnivore: An optimizer for multi-device deep learning on cpus and gpus. *CoRR*, abs/1606.04487.

[He et al., 2020] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning.

[He et al., 2015a] He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[He et al., 2015b] He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852.

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

[He et al., 2016b] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[Hendrycks et al., 2019] Hendrycks, D., Mu, N., Cubuk, E. D., Zoph, B., Gilmer, J., and Lakshminarayanan, B. (2019). Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

[Huang et al., 2017] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708.

[Huang et al., 2016a] Huang, G., Liu, Z., and Weinberger, K. Q. (2016a). Densely connected convolutional networks. *CoRR*, abs/1608.06993.

[Huang et al., 2016b] Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016b). Deep networks with stochastic depth. *CoRR*, abs/1603.09382.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[Irvin et al., 2019] Irvin, J., Rajpurkar, P., Ko, M., Yu, Y., Ciurea-Ilcus, S., Chute, C., et al. (2019). Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. *AAAI*.

[Jain et al., 1996] Jain, A. K., Mao, J., and Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *IEEE Computer*, 29(3):31–44.

[Jaiswal et al., 2021] Jaiswal, A., Babu, A. R., Zadeh, M. Z., Banerjee, D., and Makedon, F. (2021). A survey on contrastive self-supervised learning.

[Jamaludin et al., 2017] Jamaludin, A., Kadir, T., and Zisserman, A. (2017). Self-supervised learning for spinal mris. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 294–302.

[Jiao et al., 2020] Jiao, J., Droste, R., Drukker, L., Papageorghiou, A. T., and Noble, J. A. (2020). Self-supervised representation learning for ultrasound video. *Proceedings. IEEE International Symposium on Biomedical Imaging*, 2020:1847—1850.

[Johnson et al., 2016] Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L.-W. H., Feng, M., Ghassemi, M., et al. (2016). Mimic-iii, a freely accessible critical care database. *Scientific Data*, 3:160035.

[Kaushal et al., 2020] Kaushal, A., Altman, R. B., and Langlotz, C. P. (2020). Geographic distribution of us cohorts used to train deep learning algorithms. *JAMA*, 324(9):936–938.

[Kessy et al., 2018] Kessy, A., Lewin, A., and Strimmer, K. (2018). Optimal whitening and decorrelation. *The American Statistician*, 72(4):309–314.

[Kim, 2021] Kim, G. (2021). Recent deep semi-supervised learning approaches and related works. *CoRR*, abs/2106.11528.

[Kim et al., 2020] Kim, J. et al. (2020). Puzzlemix: Exploiting saliency and local statistics for optimal mixup. In *Proceedings of the International Conference on Machine Learning (ICML)*.

[Kim et al., 2021] Kim, J.-H., Choo, W., Jeong, H., and Song, H. O. (2021). Co-mixup: Saliency guided joint mixup with supermodular diversity. In *International Conference on Learning Representations (ICLR)*.

[Kitano, 2002] Kitano, H. (2002). Systems biology: A brief overview. *Science*, 295(5560):1662–1664.

[Laine and Aila, 2016] Laine, S. and Aila, T. (2016). Temporal ensembling for semi-supervised learning. *CoRR*, abs/1610.02242.

[Larsson et al., 2017] Larsson, G., Maire, M., and Shakhnarovich, G. (2017). Colorization as a proxy task for visual understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6874–6883.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Li and Fan, 2018] Li, H. and Fan, Y.-H. (2018). Non-rigid image registration using self-supervised fully convolutional networks without training data. In *Proceedings of the IEEE International Symposium on Biomedical Imaging (ISBI)*, pages 1075–1078.

[Li et al., 2020] Li, L., Zhou, X., and Wong, T. (2020). Refinement processes in machine learning. *Journal of Machine Learning Research*, 21:1–20.

[Li et al., 2022] Li, X. et al. (2022). Openmixup: A pytorch toolbox for mixup augmentations and beyond. https://github.com/Westlake-AI/openmixup. Accessed: 2024-10-06.

[Lin et al., 2017] Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. (2017). Deep gradient compression: Reducing the communication bandwidth for distributed training. *CoRR*, abs/1712.01887.

[Lipton, 2018] Lipton, Z. C. (2018). The mythos of model interpretability. *Queue*, 16(3):30.

[Litjens et al., 2017] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., van der Laak, J. A., van Ginneken, B., and Sanchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88.

[Liu et al., 2021a] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021a). Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030.

[Liu et al., 2021b] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021b). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022.

[Longo et al., 2024] Longo, L., Brcic, M., Cabitza, F., Choi, J., Confalonieri, R., Ser, J. D., Guidotti, R., Hayashi, Y., Herrera, F., Holzinger, A., Jiang, R., Khosravi, H., Lecue, F., Malgieri, G., Páez, A., Samek, W., Schneider, J., Speith, T., and Stumpf, S. (2024). Explainable artificial intelligence (xai) 2.0: A manifesto of open challenges and interdisciplinary research directions. *Information Fusion*, 106:102301.

[Loshchilov and Hutter, 2017] Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization.

[Lundberg and Lee, 2017] Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30:4765–4774.

[Mania et al., 2016] Mania, H., Pan, X., Papailiopoulos, D., Recht, B., Ramchandran, K., and Jordan, M. I. (2016). Perturbed iterate analysis for asynchronous stochastic optimization.

[McDermott et al., 2021] McDermott, M. B. A., Wang, S., Marinsek, N., Ranganath, R., Foschini, L., and Ghassemi, M. (2021). Reproducibility in machine learning for health. *Nature Biomedical Engineering*, 5(1):1–2.

[Milosheski and et al., 2023] Milosheski, M. and et al. (2023). Xai for self-supervised clustering of wireless spectrum activity. *arXiv preprint arXiv:2311.10319*.

[Moradi and Samwald, 2021] Moradi, M. and Samwald, M. (2021). Post-hoc explanation of black-box classifiers using confident itemsets. *Expert Systems with Applications*, 165:113941.

[Murphy, 2013] Murphy, K. P. (2013). *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.].

[Neghawi and Liu, 2020] Neghawi, E. and Liu, Y. (2020). Evaluation of parameter update effects in deep semi-supervised learning algorithms. *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 351–360.

[Neghawi and Liu, 2023] Neghawi, E. and Liu, Y. (2023). Analysing semi-supervised convnet model performance with computation processes. *Machine Learning and Knowledge Extraction*, 5(4):1848–1876.

[Neghawi and Liu, 2024] Neghawi, E. and Liu, Y. (2024). Enhancing self-supervised learning through explainable artificial intelligence mechanisms: A computational analysis. *Big Data and Cognitive Computing*, 8(6).

[Neghawi et al., 2023] Neghawi, E., Wang, Z., Huang, J., and Liu, Y. (2023). Linking team-level and organization-level governance in machine learning operations through explainable ai and responsible ai connector. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1223–1230.

[Newman, 2015] Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Inc.

[Oakden-Rayner, 2020] Oakden-Rayner, L. (2020). Exploring large-scale public medical image datasets. *Academic Radiology*, 27(1):147–151.

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

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.

[Pathak et al., 2016] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544.

[Perez and Wang, 2017] Perez, L. and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*.

[Pope et al., 2019] Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., and Hoffmann, H. (2019). Explainability methods for graph convolutional neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10764–10773.

[Raghupathi and Raghupathi, 2014] Raghupathi, W. and Raghupathi, V. (2014). Big data analytics in healthcare: Promise and potential. *Health Information Science and Systems*, 2(1):3.

[Ratner et al., 2017] Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. (2017). Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment*, volume 11, pages 269–282. VLDB Endowment.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). Why should i trust you? explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

[Richards, 2015] Richards, M. (2015). *Microservices: Patterns and Practices*. O'Reilly Media, Inc.

[Roberts et al., 2021] Roberts, M., Driggs, D., Thorpe, M., Gilbey, J., Yeung, M., Ursprung, S., et al. (2021). Common pitfalls and recommendations for using machine learning to detect and prognosticate for covid-19 using chest radiographs and ct scans. *Nature Machine Intelligence*, 3(3):199–217.

[Rudin, 2019] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead.

[Schwarzenberg et al., 2019] Schwarzenberg, R., Hübner, M., Harbecke, D., Alt, C., and Hennig, L. (2019). Layerwise relevance visualization in convolutional text graph classifiers. *CoRR*, abs/1909.10911.

[Shen et al., 2017] Shen, L., Margolies, L. R., Rothstein, J. H., Fluder, E., McBride, R., and Sieh, W. (2017). Deep learning to improve breast cancer detection on screening mammography. *Scientific Reports*, 7(1):244.

[Shen et al., 2021] Shen, W., Wei, Z., Huang, S., Zhang, B., Fan, J., Zhao, P., and Zhang, Q. (2021). Interpretable compositional convolutional neural networks. *CoRR*, abs/2107.04474.

[Shin et al., 2016] Shin, H., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., and Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298.

[Shorten and Khoshgoftaar, 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60.

[Shurrab and et al., 2021] Shurrab, S. and et al. (2021). Self-supervised learning methods and applications in medical imaging analysis: A survey. *arXiv preprint arXiv:2109.08685*.

[Sohn et al., 2020] Sohn, K., Berthelot, D., Li, C., Zhang, Z., Carlini, N., Cubuk, E. D., Kurakin, A., Zhang, H., and Raffel, C. (2020). Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *CoRR*, abs/2001.07685.

[Sparks et al., 2013] Sparks, E. R., Talwalkar, A., Smith, V., Kottalam, J., Pan, X., Gonzalez, J. E., Franklin, M. J., Jordan, M. I., and Kraska, T. (2013). MLI: an API for distributed machine learning. *CoRR*, abs/1310.5426.

[Szegedy et al., 2015] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.

[Taleb and et al., 2020] Taleb, A. and et al. (2020). 3d self-supervised learning for medical imaging. *Advances in Neural Information Processing Systems*, 33:18157–18168.

[Tarvainen and Valpola, 2017a] Tarvainen, A. and Valpola, H. (2017a). Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results.

[Tarvainen and Valpola, 2017b] Tarvainen, A. and Valpola, H. (2017b). Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1195–1204.

[Tarvainen and Valpola, 2017c] Tarvainen, A. and Valpola, H. (2017c). Weight-averaged consistency targets improve semi-supervised deep learning results. *CoRR*, abs/1703.01780.

[Tieleman et al., 2012] Tieleman, T., Hinton, G., et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[Uddin et al., 2021] Uddin, M. et al. (2021). Saliencymix: A saliency guided data augmentation strategy for better regularization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[van den Oord et al., 2018] van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

[Vapnik, 1995] Vapnik, V. N. (1995). *The nature of statistical learning theory*. Springer-Verlag New York, Inc.

[Vayena et al., 2018] Vayena, E., Blasimme, A., and Cohen, I. G. (2018). Machine learning in medicine: Addressing ethical challenges. *PLoS Medicine*, 15(11):e1002689.

[Wang et al., 2020] Wang, K., Wang, Y., Zhong, Z., Li, X., and Jiang, Z. (2020). Implicit semantic data augmentation for medical image classification. *arXiv*, page arXiv:2006.13940.

[Wang et al., 2019] Wang, S., Jiang, L., Shao, Z., Sun, C., and Jia, J. (2019). Implicit semantic data augmentation for deep networks. In *Advances in Neural Information Processing Systems*, volume 32, pages 12632–12641.

[Wang et al., 2021] Wang, Y., Huang, G., Song, S., Pan, X., Xia, Y., and Wu, C. (2021). Regularizing deep networks with semantic data augmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Willemink et al., 2020] Willemink, M. J., Koszek, W. A., Hardell, C., Wu, J., Fleischmann, D., Harvey, H., et al. (2020). Preparing medical imaging data for machine learning. *Radiology*, 295(1):4–15.

[Witten et al., 2016] Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[Wu et al., 2018] Wu, Z., Xiong, Y., Yu, S. X., and Lin, D. (2018). Unsupervised feature learning via non-parametric instance-level discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742.

[Xie et al., 2020] Xie, N., Ras, G., van Gerven, M., and Doran, D. (2020). Explainable deep learning: A field guide for the uninitiated. *CoRR*, abs/2004.14545.

[Xie et al., 2017] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017). Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1492–1500.

[Yang et al., 2023] Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., and Ni, B. (2023). Medmnist v2 - a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41.

[Yuan et al., 2020a] Yuan, H., Tang, J., Hu, X., and Ji, S. (2020a). XGNN: towards model-level explanations of graph neural networks. *CoRR*, abs/2006.02587.

[Yuan et al., 2020b] Yuan, H., Yu, H., Gui, S., and Ji, S. (2020b). Explainability in graph neural networks: A taxonomic survey. *CoRR*, abs/2012.15445.

[Yun et al., 2019a] Yun, S. et al. (2019a). Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[Yun et al., 2019b] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019b). Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6022–6031. IEEE.

[Zagoruyko and Komodakis, 2017] Zagoruyko, S. and Komodakis, N. (2017). Wide residual networks.

[Zaharia et al., 2012] Zaharia, M. A., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, J. M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Fast and interactive analytics over hadoop data with spark. *login Usenix Mag.*, 37.

[Zbontar et al., 2021] Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. (2021). Barlow twins: Self-supervised learning via redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10124–10133.

[Zhang et al., 2017] Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2017). Mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.

[Zhang et al., 2021a] Zhang, H. et al. (2021a). Resizemix: Mixup augmentation via resizing. *arXiv preprint arXiv:2103.05073*.

[Zhang et al., 2021b] Zhang, H., Yang, J., Gong, C., and Tao, D. (2021b). Saliency-guided mixup. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4007–4015.

[Zhang et al., 2021c] Zhang, J. et al. (2021c). Guidedmixup: Learning to improve data augmentation. *arXiv preprint arXiv:2107.10968*.

[Zhang et al., 2023] Zhang, X., Wang, Y., et al. (2023). Augmenting medical imaging: A comprehensive catalogue of 65 techniques for enhanced data analysis.

[Zhang et al., 2020] Zhang, Z., Tao, L., Zhang, Y., El-Kishky, A., and Han, J. (2020). A survey on deep learning for safety-critical autonomous systems. *arXiv preprint arXiv:2001.01264*.

[Zhao et al., 2021] Zhao, S., Hu, J., Gu, H., Wu, J., and Feng, J. (2021). Bayesian semantic data augmentation for medical image segmentation. *IEEE Transactions on Medical Imaging*, 40(10):2789–2800.

[Zhu et al., 2024] Zhu, Y., Cai, X., Wang, X., Chen, X., Yao, Y., and Fu, Z. (2024). Bsda: Bayesian random semantic data augmentation for medical image classification.

[Zintgraf et al., 2017] Zintgraf, L. M., Cohen, T. S., Adel, T., and Welling, M. (2017). Visualizing deep neural network decisions: Prediction difference analysis. *CoRR*, abs/1702.04595.