Bauhaus Universität Weimar

Faculty of Civil Engineering/Faculty of Media

Study program Digital Engineering

Self-Supervised Pre-Training For Efficient Hierarchical Image Classification: A Study On Unlabeled Data Utilization

Master's Thesis

Kushal Shah

b. 06.04.1995 in Ahmedabad

Matriculation Number: 122316

First Referee: Prof Dr. rer. nat. Tom Lahmer

Second Referee: Dr. Tilo Sperling

Submission Date: 08. March 2024

Declaration

Hereby, the author declares that, except where a specific reference of source is made, the contents and results of this thesis are his own.

Weimar, March 19, 2024.

Kushal Vikram Shah

Abstract

This master's thesis investigates the potential of self-supervised pre-training for feature extraction in hierarchical image classification tasks, focusing on the utilization of large amounts of unlabeled images. Current practices involve training separate models on limited data at each hierarchy level, leading to increased computational effort and potentially reduced performance. We propose a domain-familiar backbone Convolutional Neural Network (CNN) that can be pre-trained on a vast dataset of unlabeled images to produce high-quality embeddings, thereby reducing computational efforts across all hierarchy tiers. Our approach involves gathering and pre-processing all available unlabeled images, training a self-supervised image embedding's model such as Momentum Contrast V2 (MoCoV2), and employing this model for fine-tuning in the hierarchical image classifier. This study highlights the robustness of the trained embedding space, which is less sensitive to class distribution imbalances and can be used for image similarity comparisons to aid in labeling. The proposed method seeks to enhance the efficiency and performance of hierarchical image classification tasks by leveraging the power of self-supervised pre-training.

Keywords: Self-supervised Learning, High-Quality Embedding, Convolutional Neural Network, Unlabeled Image, Image Classifier

Acknowledgments

First and foremost I would like to thank Prof. Dr. rer. nat. Tom Lahmer for supervising and collaborating on this topic with Geberit Verwaltungs GmbH. I am grateful for his unwavering support and valuable feedback during the term of this thesis.

I would like to express my sincere gratitude to Dr. Tilo Sperling for introducing me to this challenging topic. His constant support, motivation, and guidance were incredibly valuable for the completion of my work. I would like to extend my gratitude to Mr. Marc Freibauer and Ms. Ines Saiger for their invaluable support during my thesis. They offered critical assistance and guidance when I faced major challenges. I would like to convey my gratitude to Ms. Zohour Jaouadi for her motivational support and guidance. I would also like to thank Mr. Yohannes Lisanework for his guidance and valuable feedback. I am thankful to Mr. Alexander Müller for his guidance on transfer learning principles.

Finally, I want to express my gratitude to my parents; their blessings were essential for accomplishing this task.

Contents

List of Figures

List of Tables

List of Acronyms

ANNs	Artificial Neural Networks
CNNs	Convolutional Neural Networks
MoCo	Momentum Contrast
SSL	Self-Supervised Learning
DL	Deep Learning
AI	Artificial Intelligence
ML	Machine Learning
MSE	Mean Squared Error

Acronyms not listed in here will be further explained in the text.

Chapter 1

Introduction

This thesis aims to explore the potential of self-supervised pre-training for feature extraction in hierarchical image classification tasks. The application of Convolutional Neural Network (CNN) in computer vision has achieved remarkable success but typically relies on large-scale labeled datasets for training [3]. However, obtaining labeled data in certain domains or for specific tasks can be challenging and costly. To address this, self-supervised pre-training on unlabeled images offers a promising approach. By pre-training on a large corpus of unlabeled images, high-quality embeddings can be learned that capture meaningful visual representations. This research investigates how to effectively leverage the information acquired through pre-text tasks, such as Momentum Contrast (MoCo) [4], to improve performance in downstream tasks. By leveraging the power of large amounts of unlabeled images, self-supervised pre-training has the potential to enhance hierarchical image classification by enabling the network to learn rich and generalizable visual features, contributing to improved accuracy and efficiency in complex classification tasks.

1.1 Motivation of the Thesis

The product catalog of Geberit Verwaltungs GmbH encompasses over 20,000 products organized across multiple levels of hierarchy. The current method for product identification includes an optical character recognition system which, although effective when information is visible or provided, is not always successful. Hence to address these limitations, the development of image classifiers is necessary.

Constructing an efficient traditional supervised classifier at each hierarchy is a daunting challenge for image recognition projects. Furthermore, it will require a large amount of labeled data at various levels. To mitigate the need for large amounts of labeled data and utilize readily available unlabeled data, a self-supervised machine learning approach is discussed in this thesis.

The initial phase of this research focuses on training a classifier for the first level of the hierarchical structure. This classifier is developed using a self-supervised momentum contrast pre-training technique, supplemented with a minimal amount of labeled data for classification. The results will be carefully observed, and if deemed promising, this

pre-training model will be integrated into deeper levels of hierarchy.

1.2 Thesis Outline

The structure of this thesis follows a logical progression. It begins by forming groundwork in Chapter 2 by exploring literary works in the field of artificial intelligence concerning deep learning. Moving on to Chapter 3, the focus shifts to various self-supervised learning pre-training tasks including contrastive learning, which takes center stage during the implementation phase. Chapter 4, discusses important steps related to machine learning model development in the Azure environment. It is also responsible for preprocessing image data and metadata, to make it suitable for the training pipeline during the implementation phase. The core of the thesis unfolds in Chapter 5, where the construction of the pretraining and classifier model takes place. Chapter 6, delves into the empirical investigation of the results obtained from constructed models under various setups. In the final Chapter 7, the thesis encapsulates key findings, limitations & future work.

Chapter 2

Introduction to Artificial Intelligence

"To make great products: Do machine learning like the great engineer you are, not like the great machine learning expert you aren't." [5]

Artificial Intelligence (AI) involves the development of intelligent machines that possess human-like capabilities. These machines strive to emulate human thinking and behavior by processing and analyzing data within their systems, ultimately trying to acquire human-like intelligence [6].

AI [7] is often misunderstood as some kind of modern invention but it has been around for over half a century. Alan Turing can be considered one of the earliest pioneers in the field of machine learning. In his 1947 public lecture in London, Turing expressed the need for a machine that possesses the ability to acquire knowledge through experience [8, 9]. The figure 2.1 gives a broad overview, of how deep learning is a subset of machine learning while machine learning is a subset of artificial intelligence. It is important to mention that not all machine learning techniques rely on deep learning. One example of such a technique is gradient boosting, which is widely used for constructing predictive models [10].

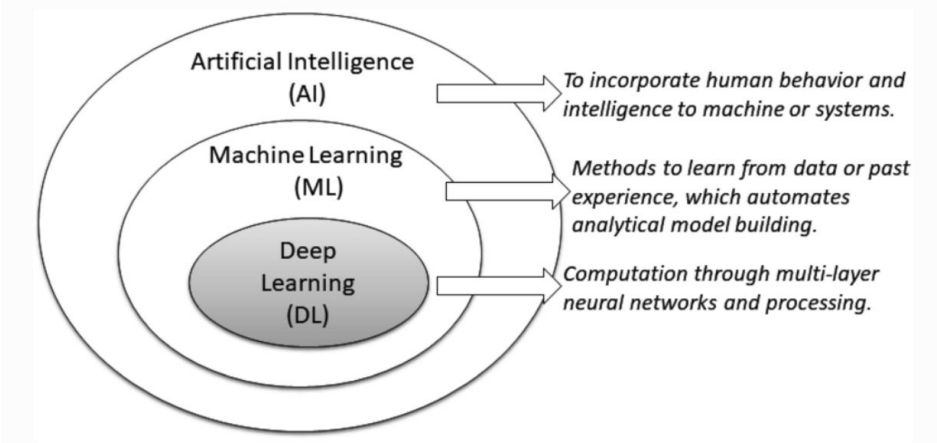

Figure 2.1: An illustration of the position of deep learning (DL), comparing with machine learning (ML) and artificial intelligence [1]

2.1 Machine Learning

Machine Learning (ML) can be described as a process in which a machine learns by itself but with some human assistance. It is particularly designed to solve tasks that require complex computations that are nearly impossible to solve by the human mind. Different ML techniques can be adopted depending on the type of problem, availability of data, and computing resources. To put it in a more concise way as described by Mitchell:

> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E ." [11]*

2.1.1 Types

Figure 2.2 gives us information on how ML can be broadly classified into supervised learning, unsupervised learning, semi-supervised learning, and reinforcement technique. The approach of each of these methods is quite different and appropriate research has to be done before choosing any of these techniques. This thesis explores the concept of self-supervised learning, which, according to Yann LeCun, Chief AI Scientist at Meta, sits in between supervised learning and unsupervised learning. It involves learning high-quality embeddings from unlabeled data while simultaneously creating pseudo-labels to learn from during training [7].

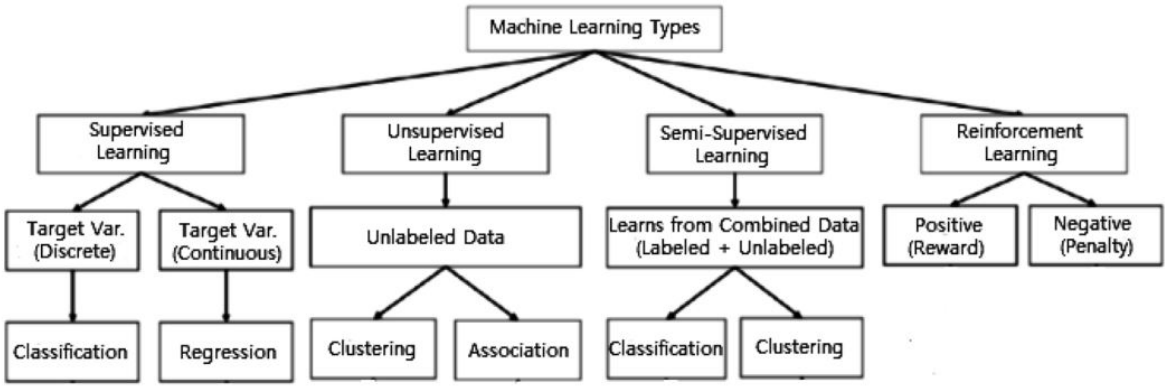

Figure 2.2: Classification of ML techniques
[12]

1. **Supervised Learning**

   Supervised Learning [13] is a machine learning approach that involves acquiring information about the input-output relationship of a system by using a predefined set of input-output training samples. Supervised learning entails the task of identifying and establishing a mapping between input data and corresponding

output data, thereby effectively supervising the outcome. In this context, supervised learning algorithms aim to discover a functional relationship, represented as a mapping function that connects an input variable (X) with an output variable (Y) by leveraging the training datasets available to them as shown in Figure 2.3 [14].

$$Y = f(X) \tag{2.1}$$

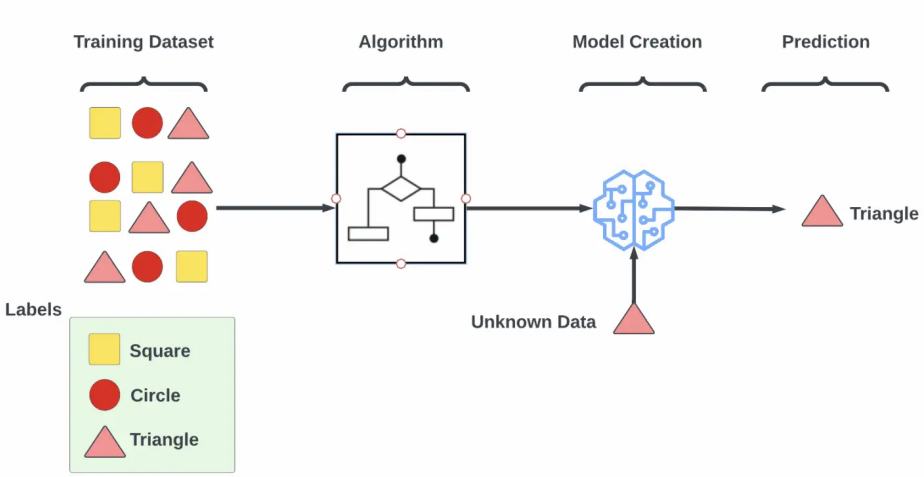

Figure 2.3: Supervised learning
[14]

2. **Unsupervised Learning**.

The principle [15, 16] behind unsupervised learning is to learn important patterns and representations from unlabeled data. The features from the data sets are grouped based on similarities and differences. Ideally, Clustering algorithms are used to sort the data as seen in Figure 2.4, into separate groups with well-defined decision boundaries. Overall, the model learns from unlabeled data or very less labeled data; this decreases the cost of the project significantly. Hence making it a very lucrative area of research.

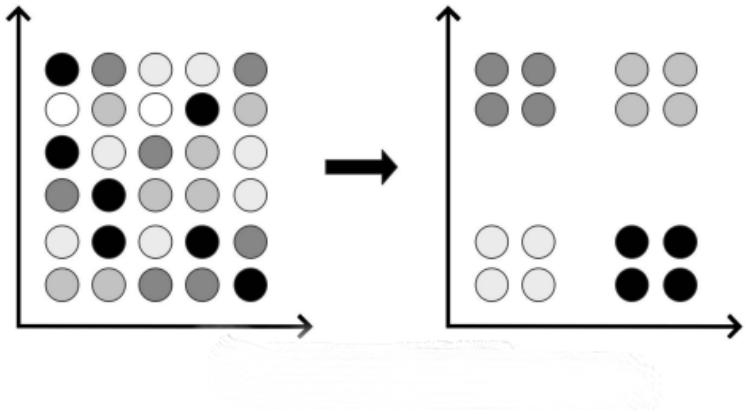

Figure 2.4: Clustering
[15]

3. **Semi-Supervised Learning**.

   Semi-supervised learning can be considered as a combination of supervised and unsupervised approaches. The goal is to reduce the caveats experienced in both methods. To achieve that, it uses a clever approach to utilize unlabeled data as well as labeled data efficiently to train the classifier [17].

4. **Reinforcement learning**.

   Reinforcement learning is inspired by experimental psychology [18]. It is based on a penalty-reward system. The algorithm can be analogous to a human teacher in a way that it receives feedback on its performance by interacting with the environment. This dynamic feedback from a changing environment enables it to learn quickly in real-time situations.

2.2 Deep Learning

Deep Learning (DL), which originated from Artificial Neural Networks (ANNs), has gained significant attention in the field of computing because of its ability to learn from data; as a result, it is widely used in computer vision and natural language processing [1, 19, 20]. The two most important deep learning methods that have a variety of applications and are important to this thesis are Artificial Neural Network (ANN) and Convolutional Neural Network (CNN).

2.2.1 Artificial Neural Network

ANNs have emerged as powerful models for various machine learning tasks, particularly in the field of deep learning. ANNs are inspired by the structure and functioning of biological [21] neural networks and are capable of learning complex patterns and relationships from input data. While ANNs are not a replica of the brain, they incorporate

certain principles inspired by neural networks found in biological systems. One key aspect is the concept of neurons. In the human brain, neurons are interconnected cells that process and transmit information through electrical and chemical signals. ANNs simulate this behavior with artificial neurons, also known as perceptrons or nodes. These nodes receive input, apply a mathematical operation to it such as a weighted sum, and produce an output signal. This is depicted in Figure 2.5. The hidden layer count can vary based on the desired features for extraction. Each neuron within these layers is activated by an activation function. This activation function helps the network to learn and discern intricate patterns. The output layer is typically configured as a fully connected softmax layer for multi-class classification in a downstream task.

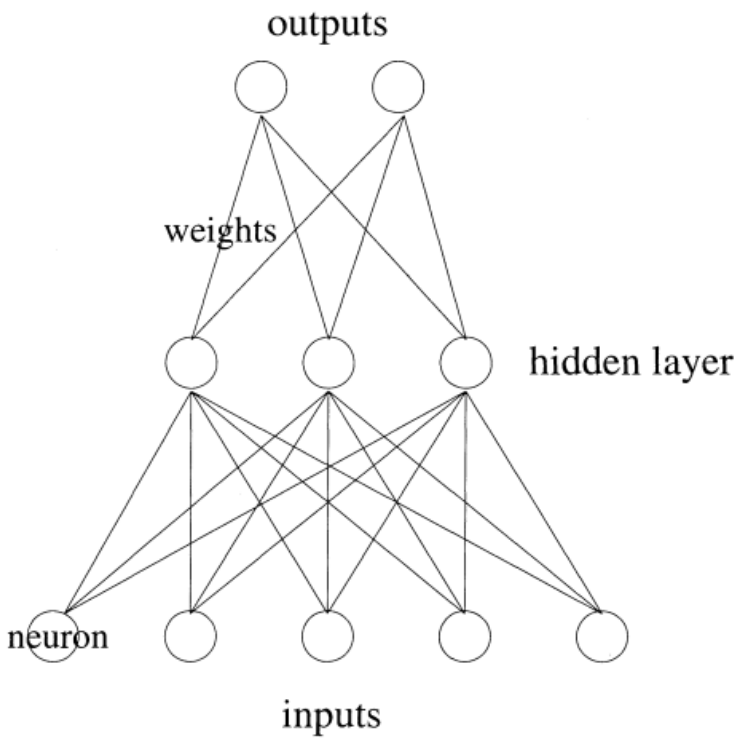

Figure 2.5: Artificial Neural network
[21]

To understand the workings of ANNs, the concept of perceptron training is pivotal and can be seen in Figure 2.6. The input or features {x_0, x_1, ..., x_n} are multiplied with their respective weights {w_0, w_1, ..., w_n} to get the weighted sum [22]. The weighted sum in conjecture to the bias of the system is processed by the activation function to generate the output. The final result represents the probability or activation level associated with a particular output or a set of outputs.

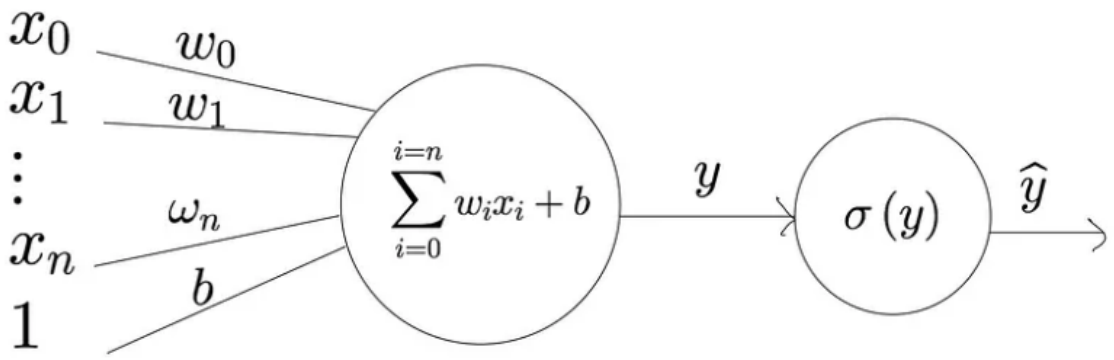

Figure 2.6: Perceptron Training
[23]

However, as discussed in section 2.1 a machine learning model learns from experience and optimizes. In ANNs or perceptron learning this is achieved by a back-propagation algorithm. During back-propagation, the network calculates the gradient of the loss function concerning each weight in the network and then adjusts the weights in the direction that minimizes the loss. This is done by derivation using a chain rule in a backward direction iteratively. In machine learning, the loss represents the difference in erroneous predictions. It tries to minimize the difference between computed and actual values, motivating the network to minimize errors during training. Hence by optimizing the loss parameter the overall performance of the network improves significantly. Mean Squared Error (MSE) is one of the most prominent used loss functions. The mathematical representation of MSE is given as [24]:

$$\text{MSE} = \frac{1}{N}\sum_{t=1}^{N}(y_t - f_t)^2 = \frac{SSE}{N}, \tag{2.2}$$

Where...

y_t is the actual outcome value at time t,

f_t is the forecast value at time t,

SSE is the sum of squared errors.

Overall, DL algorithms exhibit remarkable autonomy in the process of feature engineering and demonstrate self-learning capabilities, significantly reducing the need for human intervention [25].

2.2.2 Convolutional Neural Network

In computer vision, an image can be represented in the form of a matrix, where each value in the matrix corresponds to a pixel value in the image. Convolutional Neural Networks (CNNs) are specialized forms of neural networks designed to process, analyze, and visualize data such as images. In CNNs, an image is fed into the network as a matrix representation, where each element in the matrix corresponds to a pixel's intensity or color values. For grayscale images, a single-channel 2D matrix is used, and for colored images, a multi-channel (typically three channels for RGB) 3D matrix is employed.

The architecture of CNNs is quite different than ANNs and consists of convolutional layers, pooling layers, and fully connected layers. The convolutions are carried out by a set of kernels seen in Figure 2.7, which helps in extracting the hierarchical representation of the image which results in a distinct set of feature maps. Feature maps gradually evolve based on several hidden layers, fully connected layers, and a set of kernels. Convolutional layers are followed by sub-sampling layers or pooling layers. The purpose of pooling layers is to reduce the spatial dimensionality of the feature maps; in most cases, max-pooling or average pooling is carried out. Pooling layers ensure that there is no loss of information and introduce invariances to small translations in input data. The activation function such as ReLU or Tanh is applied element-wise after the convolutional or pooling layer to introduce the nonlinearity needed for representation learning. The activation functions serve a similar purpose as they did in ANNs. Following the convolutional layer and pooling layer fully connected layers are present in the end. They are ANNs used for decision-making. The forward pass, backpropagation, optimization algorithm, and loss function work similarly as they did in ANNs. However, the learnable parameters in CNNs are weights of kernels.

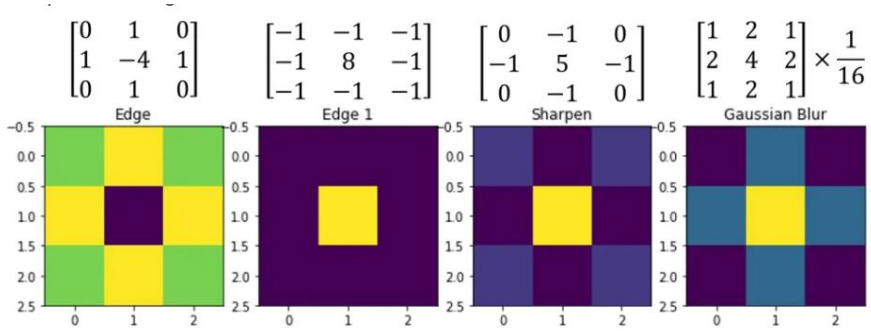

Figure 2.7: Different types of kernels used in CNN to extract representations [26]

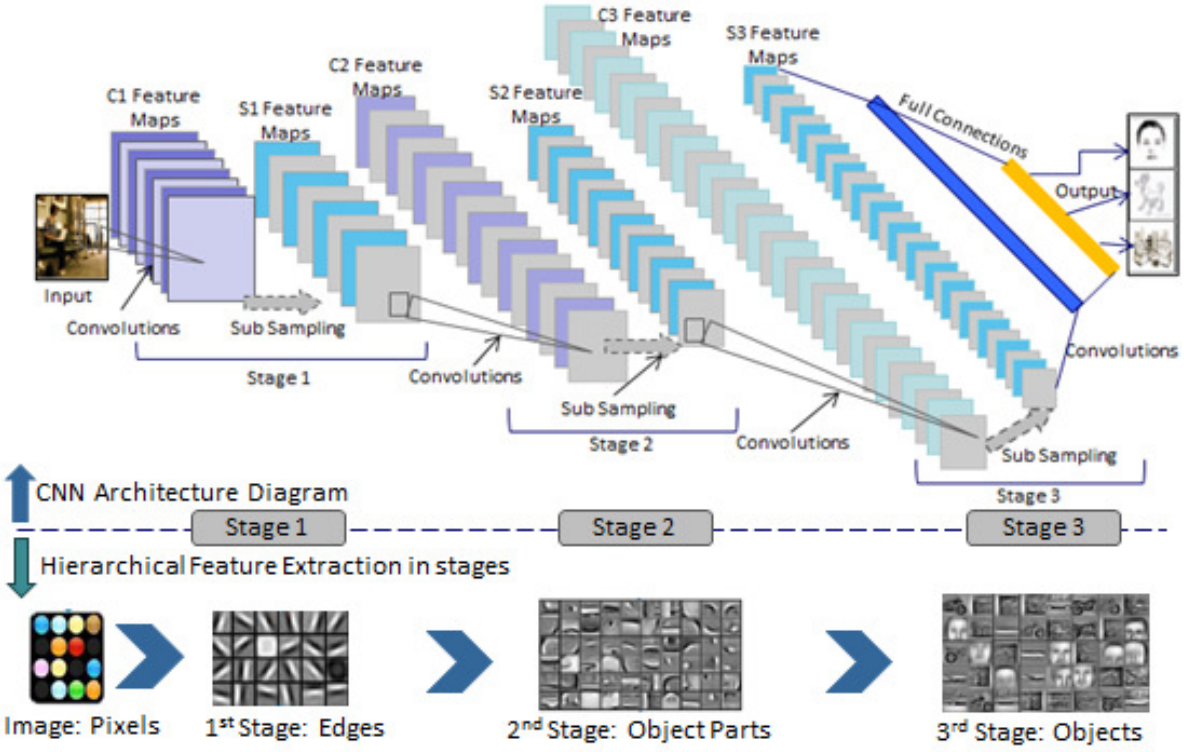

Figure 2.8: Typical CNN architecture with hidden and dense layers. As the data flows through the network, these feature maps evolve, gradually transitioning from low-level features, such as edges and textures, to high-level abstractions, such as object parts or even whole objects [27]

2.2.3 Image Classification

Image classification is the task of predicting the probabilities of correct categories of unlabeled data. As shown in Figure 2.9 a classifier algorithm is trained on the labeled dataset with a specific classification algorithm. This training process enables the algorithm to formulate rules or gain knowledge that will be useful in predicting probabilities of unknown classes.

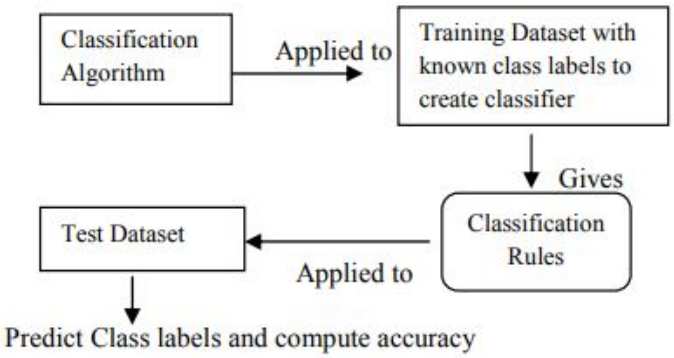

Figure 2.9: Image classification process

To evaluate a classifier, a confusion matrix is typically created with the details about correct and incorrect predictions. A confusion matrix [28] for two class classifier is shown in Figure 2.10

	Actual Positive	**Actual Negative**
Predicted Positive	True Positives (TP)	False Positives (FP)
Predicted Negative	False Negatives (FN)	True Negatives (TN)

Figure 2.10: confusion matrix

True positives (T.P) and True negatives (T.N) determine the number of examples the model was able to identify correctly. At the same time, False-Positive (F.P) and False-Negative (F.N) determine the number of misclassified examples. If the confusion matrix is extended to more classes the amount of examples in the diagonal will always show correctly identified samples. Several accuracy metrics for evaluation can be derived from this confusion matrix [28]:

- Accuracy:

  Accuracy is calculated by dividing the number of correctly classified samples by the total number of samples. In an ideal scenario where the classes are truly balanced, accuracy plays an important role.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

- Recall:

  Recall is computed as the proportion of correctly classified positive instances to the total number of actual positive instances. It can be helpful for quantitative analysis.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.4}$$

- Precision:

  Precision is determined as the ratio of correctly predicted positive values to the total number of predicted positive values. It helps to infer the quality of the model.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.5}$$

- F1 Score:

  The F1 score represents the harmonic mean between Precision and Recall scores.

$$\text{F1-Score} = \frac{2TP}{2TP + FP + FN} \tag{2.6}$$

The different metrics presented can tell the performance of the model in different aspects. The metrics for evaluation can be different based on different scenarios. For e.g. accuracy is a good metric to be evaluated when there is a balanced dataset but might not be effective when there is a high imbalance in classes. For a highly imbalanced dataset, the F1 score is a better metric for the evaluation of classification algorithm [29].

2.3 Transfer Learning

Transfer learning [30] is a machine learning technique where the knowledge of one learned task can be leveraged to tackle different, yet related tasks. For instance, the pre-trained BERT [20] model can be fine-tuned to classify various NLP downstream tasks including text classification. The core idea as illustrated in Figure [23] is to utilize a pre-trained model that has been trained on the largely available public dataset for a specific task, such as image recognition or natural language processing, to enhance generalization in another. As a result, the model is more robust to overfitting while reducing the likelihood of noise or irrelevant details in the training. This model has learned useful representations such as patterns or structures using labeled data. Now, the new model can be implemented on top of it with less or no training data. The final layers or fully connected layers are usually re-trained or replaced to suit the target task, and then fine-tuning of the model is done on a new data set relevant to that task. During fine-tuning, some layers, typically the hidden layers, have their weights "frozen" or kept fixed. This saves a lot of computational resources as well as the requirement for labeled data. As mentioned in the abstract we are using a large amount of unlabeled data; this makes transfer learning an important aspect of this thesis.

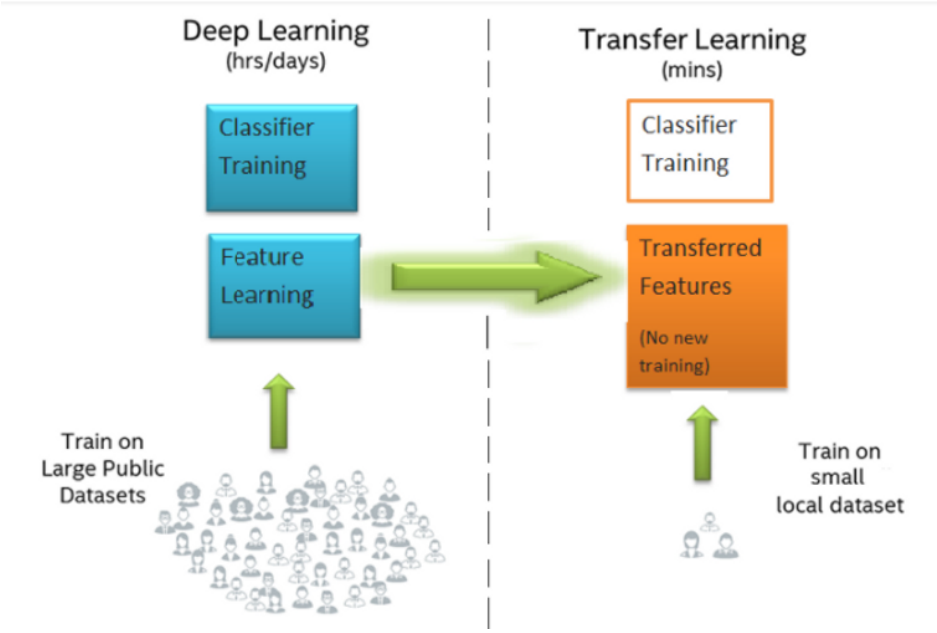

Figure 2.11: Principle of Transfer Learning w.r.t Deep Learning [23]

Chapter 3

Self-Supervised Learning

This chapter provides an in-depth exploration of the working, effectiveness, and development of Self-Supervised Learning (SSL). It provides an overview of prominent methods for extracting visual representation. Among these methods, later contrastive learning is adopted in the implementation section.

3.1 Introduction

In the realm of machine learning significant progress has been made with contrastive SSL in recent times with several studies [4, 31, 32, 33, 34] to overcome challenges posed by vast amounts of unlabeled data. These papers show us that utilizing large amounts of unlabeled data for obtaining visual representations is valuable for later downstream tasks. Commencing with the SimCLR [34], this paper establishes the cornerstone of contrastive learning, which forms the fundamental concept underpinning the extraction of representations in SSL. This foundational work offers invaluable insight into how forming negative and positive pairs can lead to the extraction of meaningful representations which is the basis of contrastive learning [34]. The breakthroughs are not only in computer vision but also extended to NLP with BERT [20] and GPT [35, 36]. All of these state-of-the-art method utilize some form of SSL in their training. After the introduction of Momentum Contrast (MoCo) [4] the gap between supervised and unsupervised learning has been reduced drastically which can be seen in Table 3.1. It was even able to surpass supervised learning in seven detection/segmentation tasks by a wide margin as depicted in Table 3.2. The compelling outcome achieved through the MoCo [4] study is the driving motivation for integrating it into Geberit's existing machine-learning pipeline for Image recognition. The study if proven successful will be able to utilize large amounts of unlabeled data generated from customer reports and would serve as a valuable resource for hierarchical image classification.

Table 3.1: Self-supervised methods such as MoCo closing gap to SL [2]

Method	Architecture	ImageNet (Top-5)		Semi-Supervised	
		Top-1*	Top-5*	1% Labels*	10% Labels*
Supervised	**ResNet50**	**76.5**	-	**56.4**	**80.4**
CPC	ResNet v2 101	48.7	73.6	-	-
InstDisc	ResNet50	56.5	-	39.2	77.4
LA	ResNet50	60.2	-	-	-
MoCo	ResNet50	60.6	-	-	-
BigBiGAN	ResNet50 (4x)	61.3	81.9	55.2	78.8
PCL	ResNet50	61.5	-	75.3	85.6
SeLa	ResNet50	61.5	84.0	-	-
PIRL	ResNet50	63.6	-	57.2	83.8
CPCv2	ResNet50	63.8	85.3	77.9	91.2
PCLv2	ResNet50	67.6	-	-	-
SimCLR	ResNet50	69.3	89.0	75.5	87.8
MoCov2	**ResNet50**	**71.1**	-	-	-
InfoMin	ResNet50	73.0	91.1	-	-
SwAV	ResNet50	75.3	-	78.5	89.9

Table 3.2: Self-supervised methods such as MoCo surpassing SL in downstream tasks [2]

Method	Architecture	Parameters	(1) Classification	(2) Detection
Supervised	AlexNet	61 M	79.9	56.8
Supervised	**ResNet50**	**25.6 M**	**87.5**	**81.3**
Inpaint	AlexNet	61 M	56.5	44.5
Color	AlexNet	61 M	65.6	46.9
BiGAN	AlexNet	61 M	60.1	46.9
NAT	AlexNet	61 M	65.3	49.4
Context	AlexNet	61 M	65.3	51.1
DeepCluster	AlexNet	61 M	72.0	55.4
Color	ResNet50	25.6 M	55.6	-
Rotation	ResNet50	25.6 M	63.9	72.5
Jigsaw	ResNet50	25.6 M	64.5	75.1
LA	ResNet50	25.6 M	69.1	-
NPID	ResNet50	25.6 M	76.6	79.1
PIRL	ResNet50	25.6 M	81.1	80.7
MoCo	**ResNet50**	**25.6 M**	-	**81.4**
SwAV	ResNet50	25.6 M	88.9	82.6

3.2 Pretext Tasks

The quest for learning meaningful representations from unlabeled has led to the development of pretext tasks in SSL. These tasks play a pivotal role in the transformation of input data into a different form that the neural network can comprehend, learn from, and try to generalize for later downstream tasks. As seen in Oleszak (2023) [4], the pretext tasks serve as the preparatory phase, with the primary aim of enabling the model to learn features also known as representations that encode the underlying structure of data. In essence, it creates the environment for SSL, by simulating a Supervised-like learning environment without the need for human intervention such as providing labeled data. Moreover, pretext tasks are versatile and can be tailored for diverse types of data, ranging from images and videos to speech and signals, among others. The design of the pretext task is based on the type of data and its domain but it can be categorized into four main categories: color transformation, geometric transformation, context-based tasks, and cross-modal-based tasks [2].

1. **Context-Based Tasks**

   The pretext task described by Noroozi and Favaro in [37] shows the effectiveness in predicting the relative positions of image patches. They have used the concept of solving a jigsaw puzzle. The pretext task as shown in the figure is where you shuffle and reorder image tiles. The task is designed as a pretext because its main purpose is not to solve puzzles but to create a supervised learning environment for the model. It provides the correct order of shuffled tiles (labeled data) without manual labeling. It can also be considered as a way to transform unlabeled data (the shuffled tiles) into labeled data (correct permutation order). While the network is trying to solve this jigsaw puzzle as seen in Figure 3.1, it learns useful representations in the process. These learned features capture intrinsic patterns and structure of the data, and after these representations are learned they can be used for other tasks, such as image classification or object detection

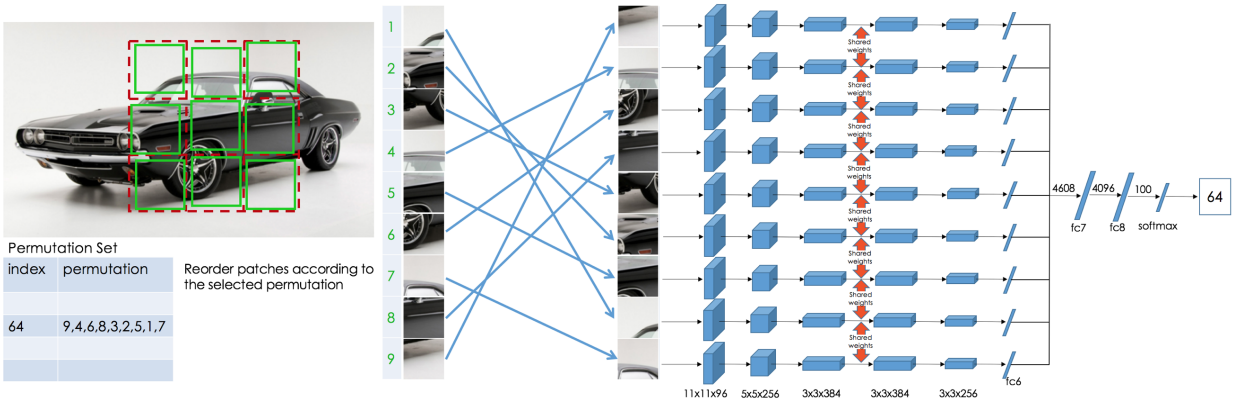

Figure 3.1: The authors tried to demonstrate the concept of jigsaw learning by taking a portion of an image, dividing it into tiles, shuffling these tiles randomly, and training a neural network to predict the correct shuffle order

[37]

2. **Colorization**

   Colorization is a pre-text task in SSL, mainly used in computer vision. The main idea behind this is to convert grayscale images into their original colored versions [38]. Zhang et al. [38] were able to achieve some outstanding results for a few cases which can be seen in Figure 3.2. While the encoder (CNNs) is trying to decipher colored images from greyscale it learns fine details about objects as well as grasps the underlying semantic information. As a result, it becomes more generalized which is helpful for further downstream tasks.

Figure 3.2: Colorization Pretext Task - Uncovering Color and Rich Semantic Understanding [38]

3. **Geometric Transformation**

Geometric transformation is a type of spatial alteration that re-arranges the pixels without changing their original pixel values [2]. These transformations encompass operations like scaling, rotating, random cropping, and flipping, both horizontally and vertically as seen in Figure 3.3. The model trained with data augmentations as a pre-text task will be more robust and scale invariant as a result. SimCLR [34] authors realized this and they used random crop, color distortion, and Gaussian blur for their data augmentation. Building on this insight, authors of MoCo [4], in their endeavor to further refine the framework, introduced MoCo V2 [33]. This updated version had a shift in data augmentation strategy while also incorporating MLP head (refer section 5).

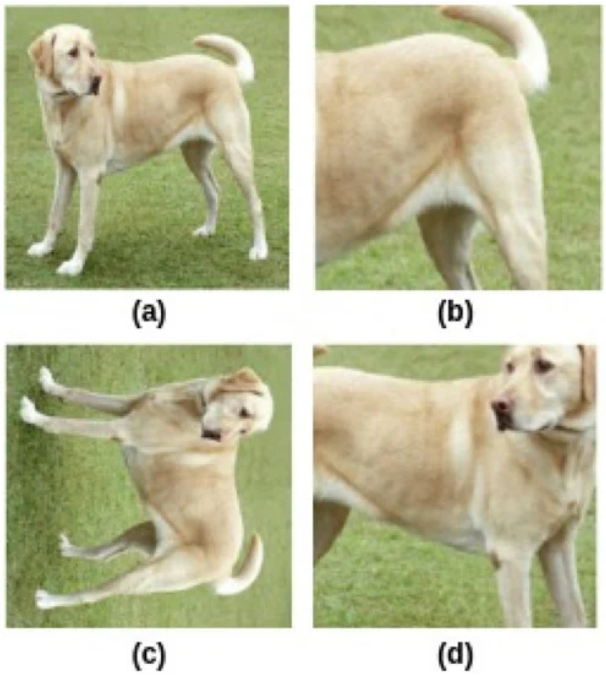

Figure 3.3: Example of geometric transformation in form of a) Original image b) Crop and resize c) Rotate d) Crop, resize, and flip
[2, 34]

4. **View Prediction (Cross-Modal-Based)**

   View prediction tasks in SSL are tailored for scenarios where multiple perspectives or angles of the same scene are present, like in video frame sequences. In a study referred to in [39], similar images from the same time sequence are considered positive pairs with anchors, while negative pairs are formed from distinct images from different times within the same sequence. So, the model learns to recognize similar features across frames from different angles as illustrated in Figure 3.4, simultaneously also distinguishing between frames that occur at different moments within the sequence.

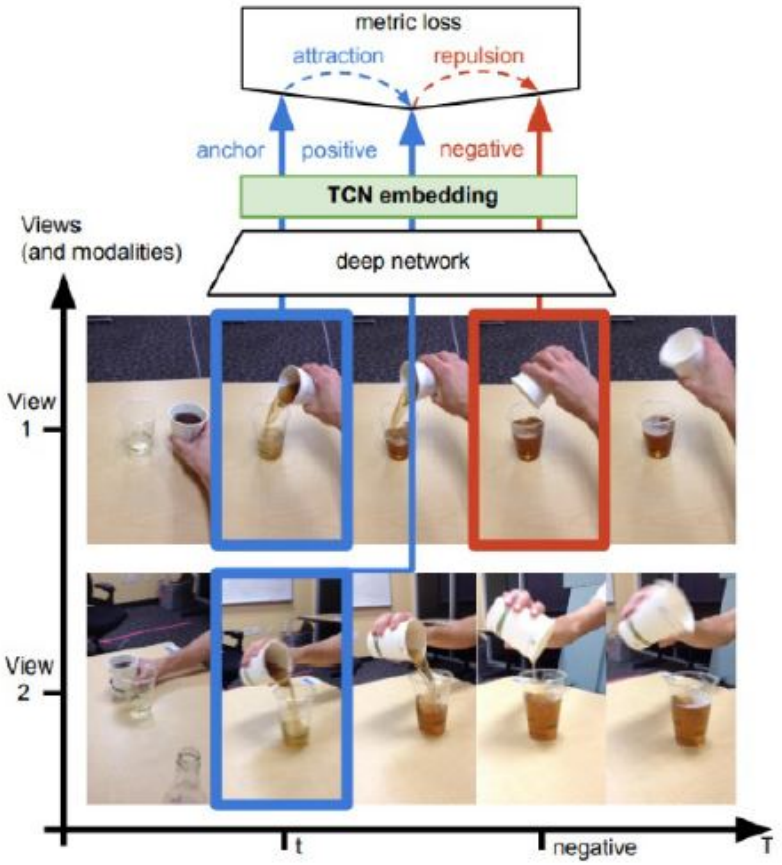

Figure 3.4: Learning representations by forming negative and positive pairs [39]

3.3 Contrastive learning

Pretext tasks as seen from the previous section serve as stepping stones, paving the way for models to develop a deeper understanding, contrastive learning just takes it one step further. It acts as the quintessential bridge between pretext tasks and robust knowledge representation as depicted in Figure 3.5. Pretext tasks help in creating initial pairs and later with contrastive learning or more specifically with contrastive loss, similar and dissimilar pairs are created. The similar samples remain closer in embedding space than those that are dissimilar hence the name contrastive learning.

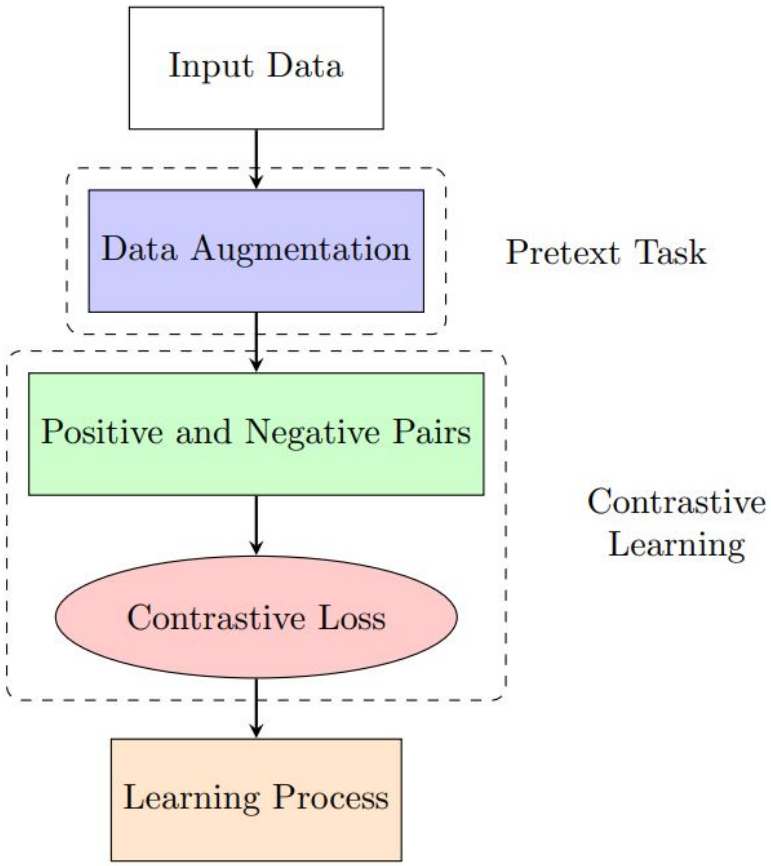

Figure 3.5: Self-Supervised Learning Transition: Pretext Task to Contrastive Learning

As seen in Figure 3.6, pairs of images from the same class exhibit proximity in the embedding space, denoted as "d+," while pairs from different classes manifest greater separation, represented as "d-." Consequently, a contrastive learning model (represented as "theta" in the example) endeavors to minimize "d+" and maximize "d-" to learn discriminative features [40].

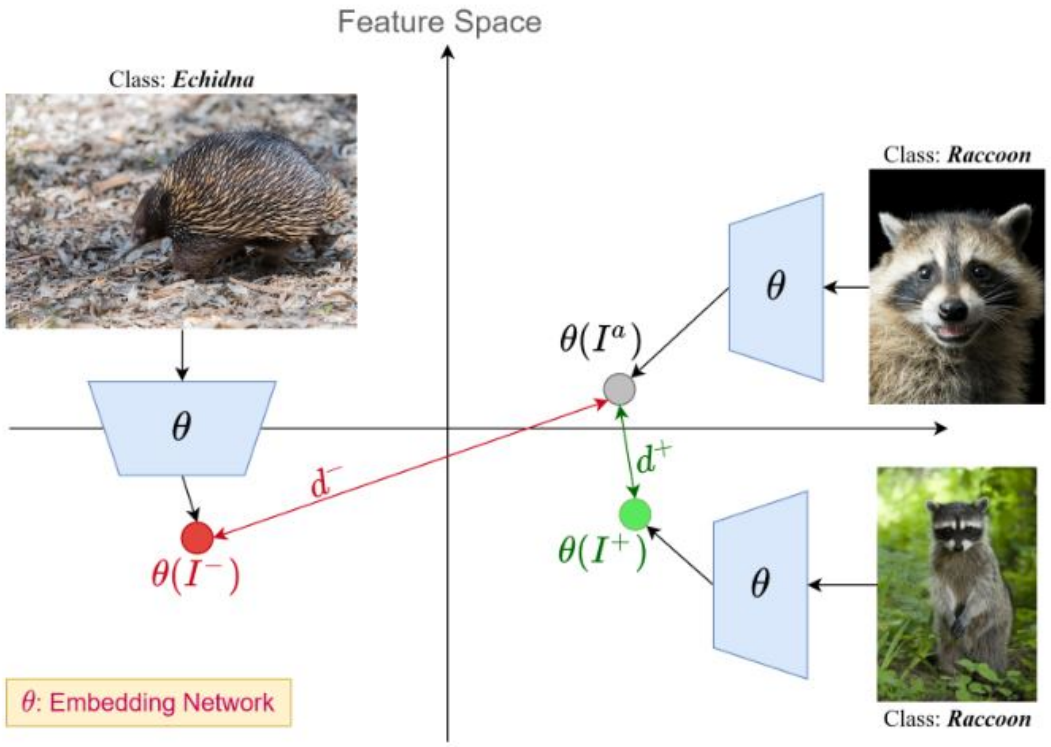

Figure 3.6: Feature Space Visualization for Similarity and Dissimilarity [40]

Chapter 4

Azure Infrastructure and Data Preprocessing

To grasp the intricacies of training a machine learning model in Azure, the initial step involved training the Momentum Contrast model on an Azure notebook using publicly available data. While the results themselves are not of paramount importance, the preparation process for training yielded valuable insights. These insights proved instrumental in effectively training the Momentum Contrast model and classifier in the later section.

4.1 Submitting a job to Azure Machine Learning Studio

The model used for pre-training MoCo uses ResNet [41][1] as the backbone architecture along with the classification head. The dataset used for training is CIFAR-10[2]. The following steps were needed to train the model successfully:

- Workspace setup

  The first step would be to create an Azure ML workspace using the Azure Portal. This will be the place where all training and execution steps of the model are saved. To establish a connection successfully to the workspace, essential details like subscription ID, resource group, etc are required.

- Environment and Script preparation

  Next in line for model training preparation involves getting the training script ready, along with all the needed files such as data and configuration. Furthermore, the establishment of a well-defined Python environment becomes imperative. To ensure a secure and seamlessly executed process, this study considers the utilization of the Docker environment, particularly with an Azure-provided image. This approach aligns with best practices for maintaining a consistent and secure computational environment in the Azure ecosystem.

[1]Resnet is a popular neural network used for deep learning techniques
[2]CIFAR is a collection of images used for training neural networks

- Registering the dataset

  The final step involves ensuring the accessibility of training data within the Azure environment. This corresponds to the mounting of local files or folders containing images as input paths during job submission.

In conclusion, this section highlighted the practical application of machine learning concepts in a cloud-based environment, emphasizing proficiency and harnessing Azure's capability for successful machine learning endeavors.

4.2 Organizing Data: Storage, Structure, and Pre-processing

This section introduces data handling and storage in the machine learning pipeline. After the brief introduction on how to build and execute the model in Azure, we will now look further at services offered by Azure for data storage and handling. The data in our use case consists of the image and its associated metadata. Different services are used to handle both. The services used in image recognition projects for data handling and storage are as follows:

1. Azure Cosmos Database
2. Label Studio Service
3. Azure blob storage

4.2.1 Azure Cosmos DB

Azure Cosmos Database [42] stands as a fully managed database solution catering to the diverse needs of modern application development, spanning AI, digital commerce, IoT, booking management, and various other solution types. It can handle NoSQL databases which means it is not restricted by any predefined fixed schema also it has a relational database feature that can allow it to be more structured when the data is needed in tabular format [43]. Also, it is scalable and can handle a growing amount of data by distributing it across multiple nodes.

The use case for Azure Cosmos Database is to store the metadata of the image. Each image has an image ID associated with it, which maps it to other services such as Label Studio and Azure blob storage. Each image is associated with metadata as described in Table 4.1. They are stored as JSON strings and organized as key-value pairs depicted in Listing (4.1).

Property	**Description**
id	Unique identifier for the image
uri_orig	Original URI of the image
file_hash	Hash value of the image file
text	Text associated with the image (null in this case)
artnr_ocr	OCR (Optical Character Recognition) results for a specific term
pii_entities	Personally Identifiable Information entities
face_rects	Rectangles specifying the locations of faces in the image (null in this case)
is_blurred	Boolean indicating whether the image is blurred or not
resolution_width	Width of the image in pixels
resolution_height	Height of the image in pixels
servicereport_artnr	Array of service report article numbers
servicereport_filename	Filename associated with the service report in PDF format
sortout	Sorting status (Keep in this case)
sortout_testset	Boolean indicating if the image is part of a test set
_rid	Resource ID
_self	Self-link of the document
_etag	Entity tag used for optimistic concurrency control
_attachments	Path to attachments
has_imagerecognition_labels	Boolean indicating whether image recognition labels are present
_ts	Timestamp

Table 4.1: JSON Property Descriptions

Listing 4.1: Meta Data of single Image


```
{
    "id": "adfe8c8ed.jpeg",
    "uri_orig": "aa2ade1ccabe0.jpeg",
    "file_hash": "5f9d04d3606",
    "text": null,
     "artnr_ocr": null,
     "pii_entities": null,
     "face_rects": null,
     "is_blurred": false,
     "resolution_width": 480,
     "resolution_height": 640,
     "servicereport_artnr": [
     "146.110.CG.1"
    ],
    "servicereport_filename": "0057.PDF",
    "sortout": "Keep",
    "sortout_testset": true,
    "_rid": "H4tAAAAAAAAAA==",
    "_self": "dbsH4t5AAcollsHA=",
    "_etag": "\"5b00216800\"",
    "_attachments": "attachments/",
    "has_imagerecognition_labels": true,
    "_ts": 1636077823
}
```


4.2.2 Label Studio Service

Label Studio, a data labeling tool available as an open-source (trial version), provides a flexible platform that caters to multiple projects, users, and data types. It serves as a versatile platform for various labeling tasks, accommodating diverse formats like audio, video, text, etc [44]. The listing shows metadata associated with each image along with its annotations. The interesting feature of Label Studio is its ability to facilitate collaborative annotation by multiple people. We can take advantage of this feature in our machine learning pipeline by making this a pipeline parameter, thus it allows us to filter data based on the count of annotators and we gain the flexibility to utilize only the annotations with a specified number of contributors. This ensures our data is highly reliable and verified.

Listing 4.2: Label Studio data


```
{
    "id": 3451943,
    "annotations": [
        {
            "id": 2120081,
            "completed_by": {
                "id": 392,
                "email": "loren_ipsum_2@geberit.com",
                "first_name": "loren",
                "last_name": "ipsum_2"
            },
            "result": [
                {
                    "id": "e9WksCSonr",
                    "type": "taxonomy",
                    "value": {
                        "taxonomy": [
                            [
                                "Badezimmersysteme",
                                "Geberit AquaClean",
                                "WC-Komplettanlagen",
                                "Wand-WCs",

                            ]
                        ]
                    },
                    "to_name": "image",
                    "from_name": "media"
                }
            ],
            "reviews": [],
            "was_cancelled": false,
            "ground_truth": false,
            "created_at": "2021-09-21T18:52:46.536860Z",
            "updated_at": "2021-09-21T18:52:46.536895Z",
            "lead_time": null,
            "prediction": {},
            "result_count": 0,
```

```
    "unique_id": null,
    "last_action": "updated",
    "task": 3451943,
    "project": 8362,
    "updated_by": 392,
    "parent_prediction": null,
    "parent_annotation": null,
    "last_created_by": null
},
{
    "id": 2120851,
    "completed_by": {
        "id": 390,
        "email": "loren_ipsum_1@geberit.com",
        "first_name": "loren",
        "last_name": "ipsum_1"
    },
    "result": [
        {
            "id": "aAd-NRZ5zk",
            "type": "taxonomy",
            "value": {
                "taxonomy": [
                    [
                        "Badezimmersysteme",
                        "Geberit AquaClean",
                        "WC-Komplettanlagen",
                        "Wand-WCs",
                    ]
                ]
            },
            "to_name": "image",
            "from_name": "media"
        }
    ],
    "reviews": [],
    "was_cancelled": false,
    "ground_truth": false,
    "created_at": "2021-09-22T11:26:54.928631Z",
```

```
        "updated_at": "2021-09-22T11:26:54.928650Z",
        "lead_time": 38.829,
        "prediction": {},
        "result_count": 0,
        "unique_id": null,
        "last_action": "updated",
        "task": 3451943,
        "project": 8362,
        "updated_by": 390,
        "parent_prediction": null,
        "parent_annotation": null,
        "last_created_by": null
    },
    {
        "id": 2307274,
        "completed_by": {
            "id": 706,
            "email": "loren_ipsum@geberit.com",
            "first_name": "loren",
            "last_name": "ipsum"
        },
        "result": [
            {
                "id": "ZDHMrr0_9T",
                "type": "taxonomy",
                "value": {
                    "taxonomy": [
                        [
                            "Badezimmersysteme",
                            "Geberit AquaClean",
                            "WC-Komplettanlagen",
                            "Wand-WCs",

                        ]
                    ]
                },
                "to_name": "image",
                "from_name": "media"
            }
```

```
            ],
            "reviews": [],
            "was_cancelled": false,
            "ground_truth": false,
            "created_at": "2021-09-29T13:24:53.548951Z",
            "updated_at": "2021-09-29T13:24:53.548974Z",
            "lead_time": 15.743,
            "prediction": {},
            "result_count": 0,
            "unique_id": null,
            "last_action": "updated",
            "task": 3451943,
            "project": 8362,
            "updated_by": 706,
            "parent_prediction": null,
            "parent_annotation": null,
            "last_created_by": null
        }
    ],
    "file_upload": "22226c40-8362_2021-09-21.json",
    "drafts": [],
    "predictions": [],
    "agreement": 100.0,
    "data": {
        "image": "https://allimagestore.blob.core.windows.net"
    },
    "meta": {},
    "created_at": "2021-09-21T17:52:42.655446Z",
    "updated_at": "2021-09-21T17:52:42.655454Z",
    "inner_id": 0,
    "total_annotations": 3,
    "cancelled_annotations": 0,
    "unresolved_comment_count": 0,

    "project": 8362,
    "updated_by": null,
    "comment_authors": []
}
```

[3]

4.2.3 Azure Blob Storage

Azure Blob Storage is Microsoft's cloud-based object storage solution, specializing in the efficient storage of vast quantities of unstructured data. The data can vary from text, binary data, images, and documents. It empowers enterprises to establish data backups as a safeguard against potential data loss. In our use case, it is used to store the image files. The images depicted in Figure 4.1 are illustrative examples of the data employed in this thesis for training and testing the uppermost hierarchical image classifier. A script was created to download images from Azure Blob Storage by establishing a connection using the connection string. The same process applies to querying the Cosmos database as well.

[3]In the Label Studio data presented here, the "completed_by" field provides details about the annotators, while the "taxonomy" field conveys information about the assigned labels.

Figure 4.1: A wide range of products can be seen from these images which makes the classification task challenging raising the need for hierarchical image classification

4.3 Data pre-processing

After gaining insights into how data is stored and managed by various services we can now delve a bit further into the data pre-processing step, which is essential for preparing the data for model implementation. So far, our setup allowed us to use labeled data. But for this thesis, we need labeled data to pre-train the MoCo model and also labeled data to train the classifier. To meet these requirements, we had to introduce a pre-processing step.

The idea was to scale the Cosmos Database structure and introduce additional JSON fields. As seen below from the pic, three new fields are now added to the container holding the metadata. These fields are ‘taxonomy values‘, ‘total annotator count‘, and ‘has_review‘. The taxonomy values are simply the annotation’s value from label studio export as shown in Listing (4.2). As a result, connecting Label Studio export in the pipeline is not required to get the annotation detail. Annotator count specifies the number of annotators that annotated the image and the ‘has_review‘ field is an extra validation step where an expert reviews the label of an image which is a boolean value.

Listing 4.3: CosmosDB data with additional fields

```
{
    'id': 'adfe8c88-8110-4591-9b73-55cd702673ed.jpeg',
    % old schema
    % new properties
    'labels_with_taxonomy': [
        {'annotator_id': 706, 'taxonomy': [
            ['Badezimmersysteme',
            'Betatigungsplatten und WC-Steuerungen',
            'Betatigungsplatten',
            'Fur Twinline Unterputzspulkasten']
        ]},
        {'annotator_id': 709, 'taxonomy': [
            ['Badezimmersysteme',
            'Betatigungsplatten und WC-Steuerungen',
            'Fur Twinline Unterputzspulkasten',
            'Spul-Stopp-Spulung'],
            ['Badezimmersysteme',
            'Geberit AquaClean',
            'WC-Aufsatze',
            'Geberit AquaClean 5000',
            'Geberit AquaClean 5000 WC-Aufsatz']
        ]},
        {'annotator_id': 1662, 'taxonomy': [
            ['Badezimmersysteme',
            'Geberit AquaClean',
            'WC-Aufsatze',
            'Geberit AquaClean 5000',
            'Geberit AquaClean 5000plus WC-Aufsatz']
        ]}
    ],
    'total_annotations': 3,
    'has_review': True
}
```

The task was challenging because each image has multiple annotators specified by unique annotator IDs as seen in the Label Studio export referenced in Listing 4.2. Each of them has its annotations, possibly one or more. To unify them in a useful way a JSON property was achieved with a script. Now that we have labeling information for all the images we can just query the Cosmos Database with the specific labels or choose

unlabelled data as well which was not possible previously. The flexibility gained with this was necessary for further steps in setting up the model

We managed to simplify a step further by including specific choices of labels that we wish to include and exclude in the label taxonomy. So now there is the possibility to query the database with specific taxonomy information as shown in Figure 4.2 or get only unlabeled data as well. This is done by creating a user-defined function in the Cosmos Database. The logic in the function takes care of filtering the data to our needs.

```
    query = f"""

SELECT c.id, udf.filterDocumentsByLabels(c.id, c.labels_with_taxonomy,
['include_label_list'], ['exclude_label_list']) as Annotations
FROM c
WHERE c.is_blurred = false
AND
ARRAY_LENGTH(udf.filterDocumentsByLabels(c.id, c.labels_with_taxonomy,
['include_label_list'], ['exclude_label_list'])) > 0

"""
```

Figure 4.2: Example of how user-defined function can be used to filter data

Having concluded the pre-processing steps along with a foundational understanding of Azure, the upcoming section will give insights into the implementation stage, connecting the groundwork laid to the practical execution of the project.

Chapter 5

Implementation

The implementation phase stands out as an intricate aspect of the thesis, which involves primarily creating of Momentum contrast model and modifying the existing classifier model along with the application of judicious transfer learning methodologies discussed in section 5.1.

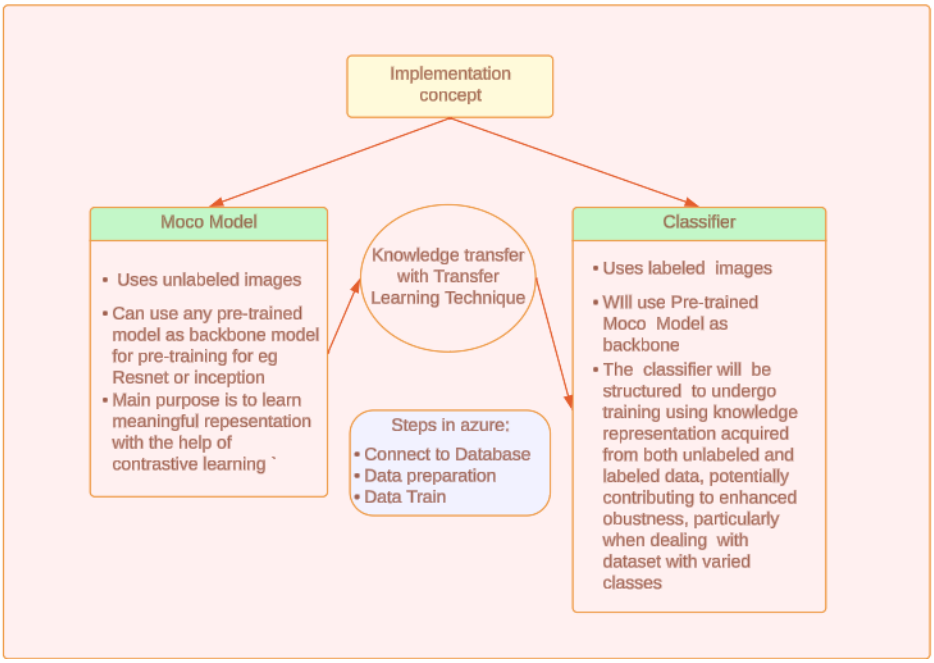

Figure 5.1: Implementation flow chart

5.1 Momentum Contrast Model

The momentum contrast model [4, 33] created uses unlabeled data for learning meaningful representations with the help of contrastive learning [34]. It can be viewed as a dynamic dictionary [4, 33] look-up process by integration of a queue and moving-averaged encoder. The dynamic dictionary created on the fly with these encoders provides an evolving source of negative images crucial for contrastive learning [34] as well as providing the stability to model with a moving average encoder enabling the model to learn intricate visual patterns without too much dependence on labeled data [4, 33]. In rather simple terms, MoCo [4, 33] is a technique enabling a machine learning model to glean insights from unlabeled images by constructing a visual pattern ”dictionary” and em-

ploying an intelligent approach to image comparison for learning. Figure 5.2 illustrates the steps involved in creating the MoCo V2 [33] model and its detailed implementation is discussed in model training 5.1.3.

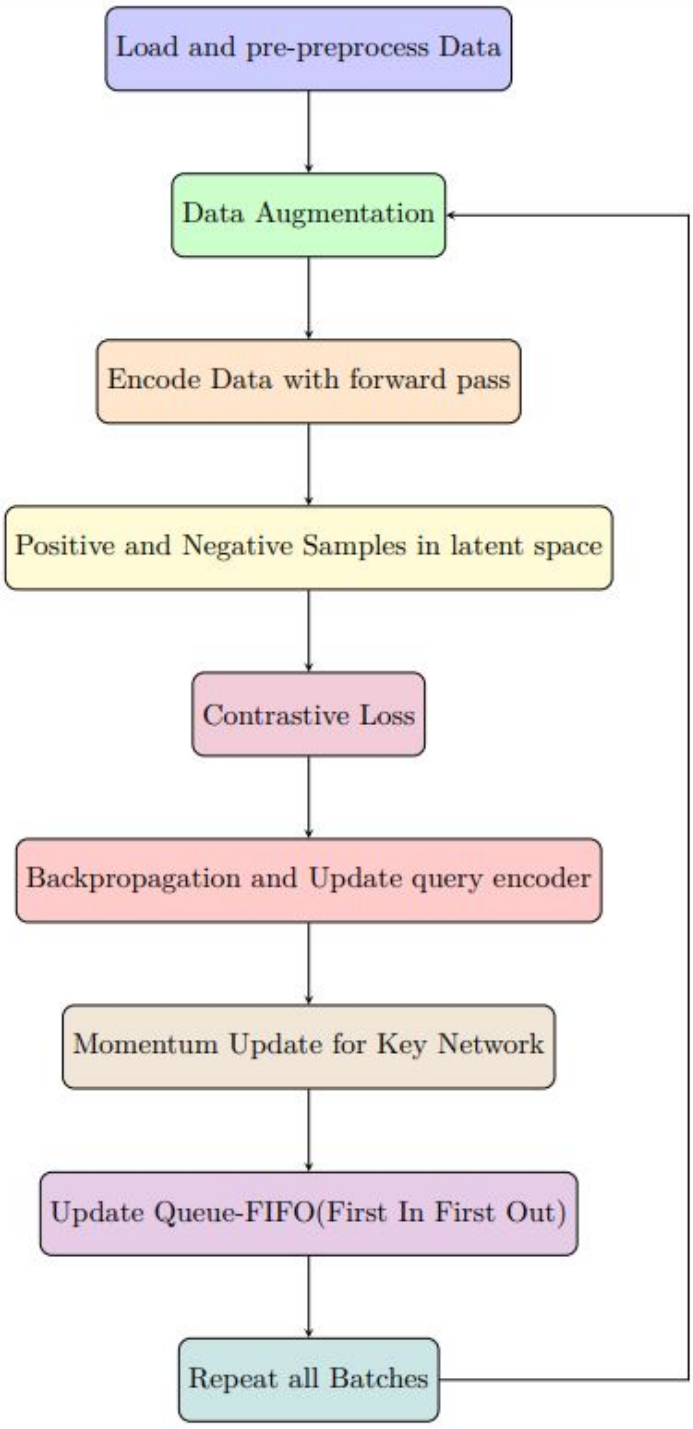

Figure 5.2: Momentum Contrast flow chart

The actual implementation aligns with the three primary stages in Azure Machine Learning Studio:

1. Connecting to Database (‘**connect _db_MoCo**‘)
2. Data preparation (‘**data_prep_MoCo**‘)
3. Model training (‘**data_train**‘)

To orchestrate the flow of the model for these steps an execute file is created. It also manages pipeline parameters, authentication, configuration settings, and the creation of a docker container for setting up the Python environment for the machine learning job.

5.1.1 Connecting to Database (‘connect _db‘)

This module or script plays an important role in preparing unlabeled dataset effectively by connecting to Azure Cosmos Database. With the help of command line arguments, a custom query for the Cosmos database (section 4.2.1) is passed as a pipeline parameter from the execute file. Leveraging the power of query as described in section (4.3), the script efficiently filters unlabeled Image ID data from the Cosmos Database, creating a structured dataset using Pandas Data Frame. The output of this module is a training set containing Image IDs as a CSV file but also generates a JSON file containing the mapping of image filenames to their respective Cosmos DB container.

In addition to its core functionalities, the script provides detailed logging, offering insights into the dataset's composition, the number of images in the training set, and the paths where the data is stored. This logging capability enhances transparency and facilitates the monitoring of dataset characteristics as well as helps in the debugging process. Collectively, as the initial step in the machine learning pipeline, this script stands as an effective tool for streamlining the extraction, curation, and organization of unlabeled dataset.

5.1.2 Data preparation (‘data_prep_moco‘)

The output of **‘connect_db‘** is essentially input for **‘data_prep‘** step. This script as shown in Figure 5.3 downloads and pre-processes images from an Azure Blob Storage container (4.2.3) which will be used in the training step. To begin with, it has **’load_data function’** which reads CSV file containing image IDs from the previous step, and **’get_image_size’** function which determines the image size required for model training based on the PyTorch model. Furthermore, more **’get_image’** function asynchronously downloads and resizes images from Azure Blob Storage.

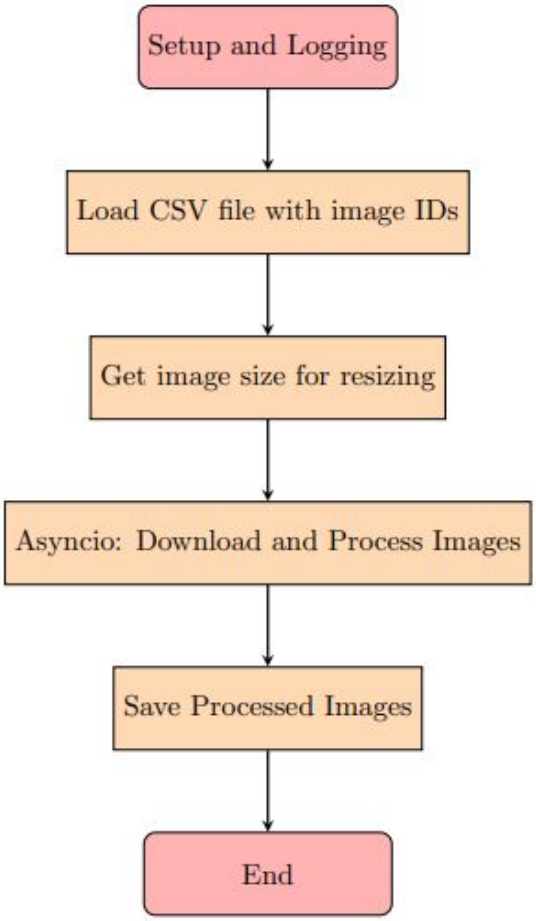

Figure 5.3: Data preparation steps

5.1.3 Model training ('data_train')

After data preparation, this module focuses on training and building of MoCo V2 [33] model as shown in Figure 5.2. The workflow is segmented into three distinct stages, namely:

1. Data loading and Transformation **('datamodule_MoCo')**
2. Momentum Contrast model **('MoCo_model')**
3. Encapsulating and saving the model **('training_MoCo_pipeline')**

1. Data loading and Transformation **('datamodule_moco')**

   The Data module script handles the loading of images from **'data_prep'** step and prepares batches of images after MoCo V2 [33, 34] transformations. These transformations are applied randomly to each image in the dataset. The following transformations are included:

- **Crop size**: The crop size of the Image is defined after Resizing. By cropping it allows the model to learn the variability by training different views [34] of image in every epoch.
- **Color Jittering**: Adjusts the brightness, contrast, saturation, and hue of images.
- **Grayscale**: Randomly converts images to grayscale.
- **Gaussian Blur**: Randomly applies Gaussian blur to the images.
- **Resizing**: Resizing the images to introduce variability in scale.
- **Horizontal Flip**: Randomly flips images horizontally.
- **Vertical Flip**: Randomly flips images vertically.
- **Rotation**: Randomly rotates images within specified degrees

After all the random transformations are performed, the normalization of images is done using mean and standard deviation values obtained from the backbone model's default configuration. This ensures pixel values are in a standard range to facilitate convergence during training. It is also important to note that certain transformations applied during the pretraining phase may vary from those typically employed in the classifier model.

For each image, two transformed versions of the same image are obtained as the output of MoCo V2 transforms [33, 34]. They are known as a query image and a key image [4]. These pairs represent positive samples for contrastive learning while the negative images are the remaining images in the batch [34].

2. Momentum Contrast model **('moco_model')**

The MoCo V2 [33] model is based on a ResNet [41] backbone with the classification head removed. The current pipeline is suitable for most of the ResNet-based [41] architecture. To reduce computational costs and simplicity the most basic model i.e. ResNet18 [41] is used as the backbone in all experiments. These models are readily available in Pytorch libraries with an option to freeze their weights or train them again.

To understand MoCo V2 [33] algorithm 2, it is important to understand contrastive loss and data augmentation practices in the SimCLR [34] framework. The significance arises as authors of MoCo V2 adapted strong augmentations and Multilayer projection head concept from SimCLR to make it better than the initial MoCo [4] and SimCLR frameworks.

Contrastive Learning Framework in SimCLR:

The contrastive learning algorithm 1 proposed in SimCLR [34] learns visual representation by emphasizing the consistency among variously augmented views of identical data samples through a contrastive loss within the latent space as shown in Figure 5.4 and Figure 5.5.

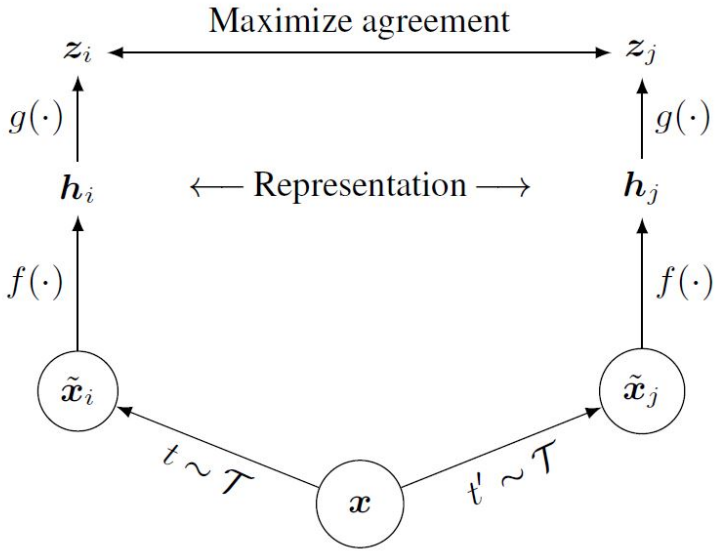

Figure 5.4: visual representation of SimCLR Algorithm [34]

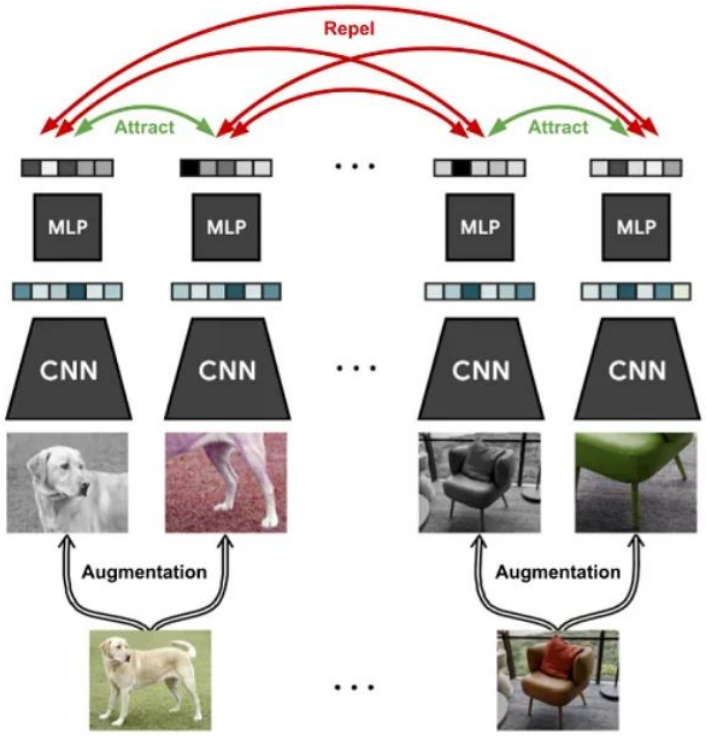

Figure 5.5: Illustrative example of SimCLR framework [45]

Algorithm 1 SimCLR's Main Learning Algorithm [34]

Require: Batch size N, constant τ, structure of f, g, T

for $k = 1$ to N **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    Draw two augmentation functions $t \sim T$, $t_0 \sim T$
    $x_{2k-1} \leftarrow t(x_k)$ ▷ The first augmentation
    $h_{2k-1} \leftarrow f(x_{2k-1})$ ▷ Representation
    $z_{2k-1} \leftarrow g(h_{2k-1})$ ▷ Projection
    $x_{2k} \leftarrow t_0(x_k)$ ▷ The second augmentation
    $h_{2k} \leftarrow f(x_{2k})$ ▷ Representation
    $z_{2k} \leftarrow g(h_{2k})$ ▷ Projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \frac{z_i^T z_j}{\|z_i\| \|z_j\|}$ ▷ Pairwise similarity
  end for
  Define $\ell(i, j)$ as $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} 1[k \neq i] \exp(s_{i,k}/\tau)}$
  $L = \frac{1}{2N} \sum_{k=1}^{2N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  Update networks f and g to minimize L
end for
return Encoder network $f(\cdot)$, and discard $g(\cdot)$

As illustrated in Figure 5.4, the framework consists of four main components:

(a) Stochastic Data Transformations:

Given an image from the dataset, it undergoes transformations and forms two augmented versions of a single image Figure 5.4. It goes transformations such as random crop, resize, and Gaussian blur which forms $\tilde{x}_i$ and $\tilde{x}_j$ as positive pairs. Similar transformations can be seen in the MoCo V2 model as well.

(b) Base Encoder f(.)

The backbone model, often referred to as the base encoder is responsible for extracting representations from augmented versions of the same image. During the extraction process, the image is passed to the base encoder (neural network) consisting of various layers and parameters that have been trained to understand and capture relevant features within an image (section 2.2.2). As the augmented version of the image is passed through the network, it produces a numerical representation, often called feature vector or embedding, in high dimensional space. It can be regarded as a compressed and abstract

form of image. The representation [34] is given by:

$$h_i = f(\tilde{x}_i) = \text{ResNet}(\tilde{x}_i). \tag{5.1}$$

(c) Neural Network Projection Head g(.)

The representations from the base encoder are not directly involved in calculating contrastive loss, instead, it is passed to the projection head. The projection head consists of one hidden layer with multi-layer perceptrons and its output is obtained through this equation[34]:

$$z_i = g(h_i) = W^{(2)}\sigma(W^{(1)}h_i), \quad \text{where } \sigma \text{ is ReLU (Non-linearity).} \tag{5.2}$$

The introduction of non-linearity allows the model to capture complex patterns and relationships within the data that might not be possible through a linear transformation alone in the base encoder. It might limit the model's ability to capture complex patterns and hierarchical features in the dataset. Studies conducted in SimCLR [34] prove that representation after the projection head is advantageous to calculate contrastive loss rather than using the projections directly from the base encoder.

(d) Contrastive Loss

After the projections (tensors) in latent space, it would be possible to calculate contrastive loss. Consider a random sample of batch size N as depicted in the algorithm 1; it is selected, and by applying augmentations, we obtain 2N data points. The negative samples are not explicitly calculated here; instead, for a positive pair, the remaining 2(N-1) augmented examples within a sample mini-batch serve as negative examples [34]. The loss function is expressed as:

$$\begin{aligned}
\mathcal{L}(i,j) &= -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} 1[k \neq i] \exp(\text{sim}(z_i, z_k)/\tau)}, \\
Z_i &= \text{augmented version of image sample}, \\
Z_j &= \text{second augmented version of the same image sample}, \\
Z_k &= \text{Data points from other images in the mini-batch}(k \neq i), \\
T &= \text{Temperature parameter for smoothing of´function}.
\end{aligned} \tag{5.3}$$

Normalized temperature-scaled cross-entropy (NT-Xent) [46] loss is used for contrastive loss, which essentially computes cosine similarity (sim(zi,zj), sim(zi,zk)) for both the numerator and denominator. The objective is to simultaneously maximize the numerator, representing positive pairs' similarity,

and minimize the denominator, reflecting dissimilarity with other samples in the mini-batch, thereby ensuring the minimization of the negative log term and the decrease of overall contrastive loss across all pairs. As a result, similar pairs will be paired together in latent space, and negative pairs are pushed apart, which helps create better decision boundaries for downstream tasks [34].

The downside of this end-to-end mechanism [4] is that negative samples are dependent on batch size. Therefore to make the model more robust it needs a high batch size (4k-8k). This makes it computationally expensive to train the model.

To overcome this challenge authors of MoCo proposed building a dynamic dictionary on the fly for the key encoder to maintain negative keys as shown in Figure (5.6). This algorithm 2 fundamentally differs from SimCLR explained above in two ways: Building a dynamic dictionary and incorporating the momentum encoder.

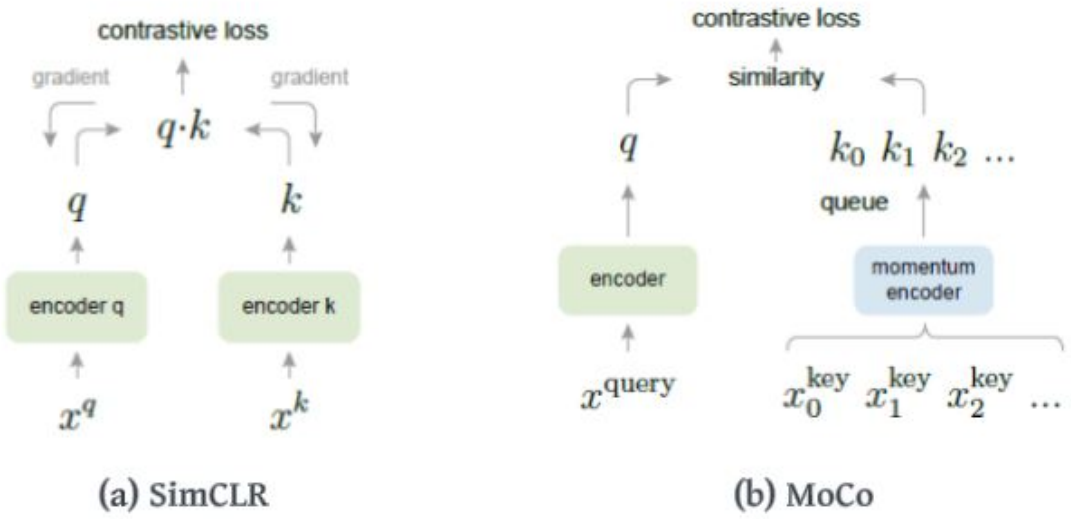

Figure 5.6: Both encoders are updated by gradient descent in (a), while MoCo(b) uses momentum encoder
[4]

Algorithm 2 Momentum Contrast Algorithm. [4]

Input: f_q, f_k (encoder networks for query and key), *queue* (dictionary as a queue of K keys), m (momentum), t (temperature)

f_k.params = f_q.params ▷ Initialize
for x **in** *loader* **do** ▷ Load a minibatch x with N samples
  $x_q = \text{aug}(x)$ ▷ Randomly augmented version for query
  $x_k = \text{aug}(x)$ ▷ Randomly augmented version for key
  $q = f_q.\text{forward}(x_q)$ ▷ Queries: $N \times C$
  $k = f_k.\text{forward}(x_k)$ ▷ Keys: $N \times C$
  $k = k.\text{detach}()$ ▷ No gradient to keys
  $l_{\text{pos}} = \text{bmm}(q.\text{view}(N, 1, C), k.\text{view}(N, C, 1))$ ▷ Positive logits: $N \times 1$
  $l_{\text{neg}} = \text{mm}(q.\text{view}(N, C), queue.\text{view}(C, K))$ ▷ Negative logits: $N \times K$
  $logits = \text{cat}([l_{\text{pos}}, l_{\text{neg}}], \text{dim} = 1)$
  $labels = \text{zeros}(N)$
  $loss = \text{CrossEntropyLoss}(logits/t, labels)$ ▷ Contrastive loss
  $loss$.backward() ▷ SGD update: query network
  update(f_q.params)
  f_k.params = $m \times f_k$.params + $(1 - m) \times f_q$.params ▷ Momentum update: key network
  enqueue($queue, k$) ▷ Enqueue the current minibatch
  dequeue($queue$) ▷ Dequeue the earliest minibatch
end for

Abbreviations: `bmm`: batch matrix multiplication; `mm`: matrix multiplication; `cat`: concatenation.

Dynamic dictionary:

The idea is to maintain a dictionary of keys, which can be treated as a queue of data samples. This configuration allows for the reuse of encoded keys from the recent mini-batches, effectively disentangling the size of the dictionary from that of the mini-batches. To update this dictionary on the fly, the oldest mini-batch is removed to add the latest batch, ensuring the dictionary maintains the sampled subset of the entire dataset [4]. It works on the First In First Out principle. Furthermore, the dictionary/queue could be set up as a flexible hyperparameter. The other advantages include reduced overhead in maintaining a dictionary compared to higher batch sizes coupled with the replacement of the oldest keys which aligns less consistently with new ones.

Momentum Encoder:

As seen in the SimCLR algorithm 1, both the encoders are updated by backpropagation but it is different in the case of MoCo as seen in Figure 5.6. The gradients for only the query encoder are updated with backpropagation but to update the gradients of the key encoder it has to render across all samples of the queue (dynamic dictionary) making it an impractical task. Hence the authors have come up with the concept of momentum encoder also known as average encoder to update the gradients of the key encoder. Let's denote the query and key encoders as f_q and f_k, and their parameters as θ_q and, θ_k respectively. Here is how the gradients of the key encoder are updated:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q \tag{5.4}$$

The above equation indicates that the parameters of the key encoder are dependent on the query encoder. For e.g, if we make m=0.99 and use this value in the equation 5.4, it would be $0.99\theta_k$+$0.01\theta_q$. In order to update θ_k we retain 99% of existing θ_k and replace 1% information from θ_q. As a result, the parameters of the key encoder are evolving gradually and becoming consistent with the query encoder. Studies [4] have shown the higher the value of m, the slower the evolution of parameters of key, which will be able to capture consistent representation over the entire training process

The contrastive loss in MoCo and MoCo V2 is similar and is known as InfoCE[47] loss, it is similar to NT-Xent [46] loss except the similarity measure used is dot product rather than cosine similarity. To summarize, the MoCo V2 algorithm consists of the MoCo algorithm as a base, with improvements adapted from SimCLR such as stronger augmentations and MLP head. MoCo also has other hyperparameters such as learning rate and weight decay. Weight decay is often used as a regularization parameter. The implementation in script '**moco_model**' consists of the MoCo V2 model. For further downstream tasks such as image classification, the trained query encoder is attached to fully connected layers with a softmax function to predict classes. This is discussed in the classifier model (Section 5.2).

3. Encapsulating and saving the model **('training_moco_pipeline')**

This script serves as an entry point of the training process. It specifies the input path for data to be used by the MoCo model and configures training epochs. Monitoring the training progress and defining key metrics, such as contrastive loss, is handled within this script. Additionally, it establishes the output path and saves the model after training. The MoCo model created is saved in the Azure model

store which can be later used as the base model for the classifier. The output folder contains a checkpoint file with details about the weights and parameters of each layer. The plot for contrastive loss can also be found here.

5.2 Classifier Model

The classifier model serves as the concluding component of the implementation phase. The learned representations from the pre-trained MoCo [4] model are passed to this model with the help of transfer learning (section 2.3) for the further classification process. It consists of fully connected layers for the downstream classification task. The goal is to make the first-level classifier for the following categories:

1. Badezimmer Systeme
2. Wekzeuge und Netzwerkkomponenten
3. Sort Out
4. Installations-und Spülsysteme

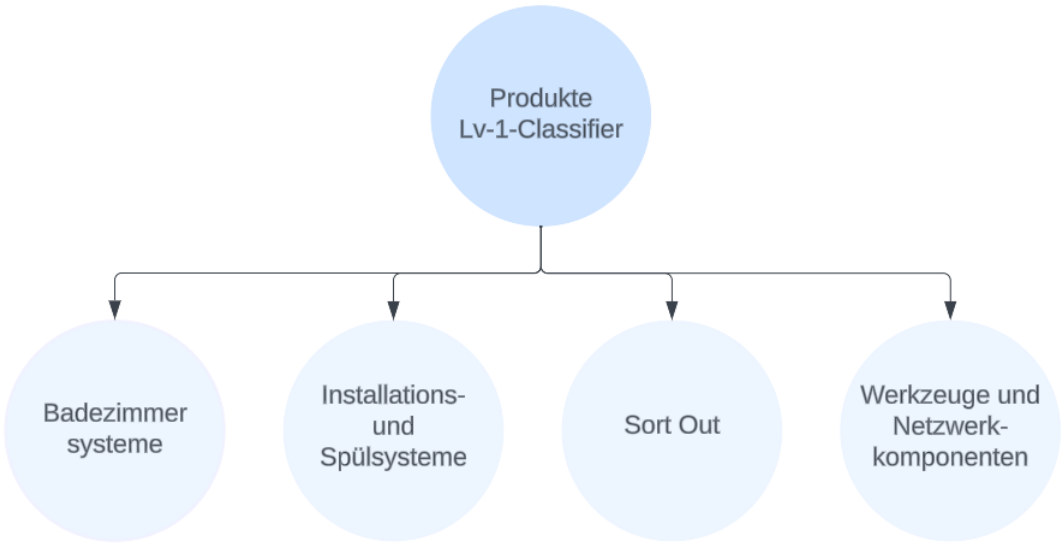

Figure 5.7: Prediction Classes

It is important to note that 'Sort-Out' is not any product in the Geberit catalog but these are images with no useful information, hence it is better to sort them out in the first level. The important steps in the classifier model can be aligned with Azure implementation as follows:

1. Connecting to Database ('**connect _db**')
2. Data preparation ('**data_prep**')

3. Model training ('**data_train**')

5.2.1 Connecting to Database ('connect _db')

The connect DB works similarly to that explained in section 5.1.1. The only difference is that it now queries labeled data for the categories mentioned in Figure 5.7. Furthermore, this step splits the data for training, testing, and validation by creating CSV files consisting of Image ID along with its labels which can be seen below in Figure 5.11.

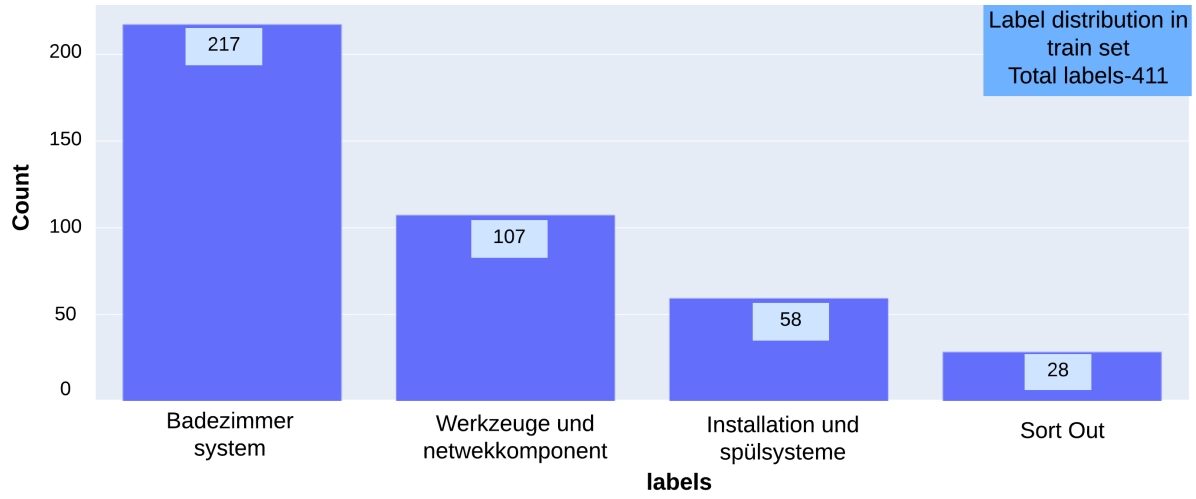

Figure 5.8: Train Set

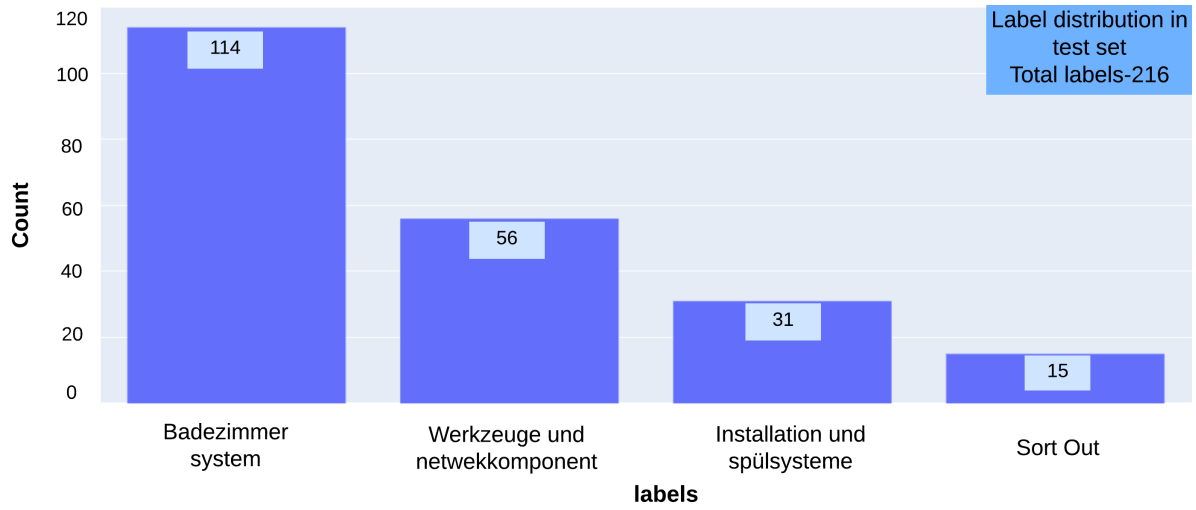

Figure 5.9: Test Set

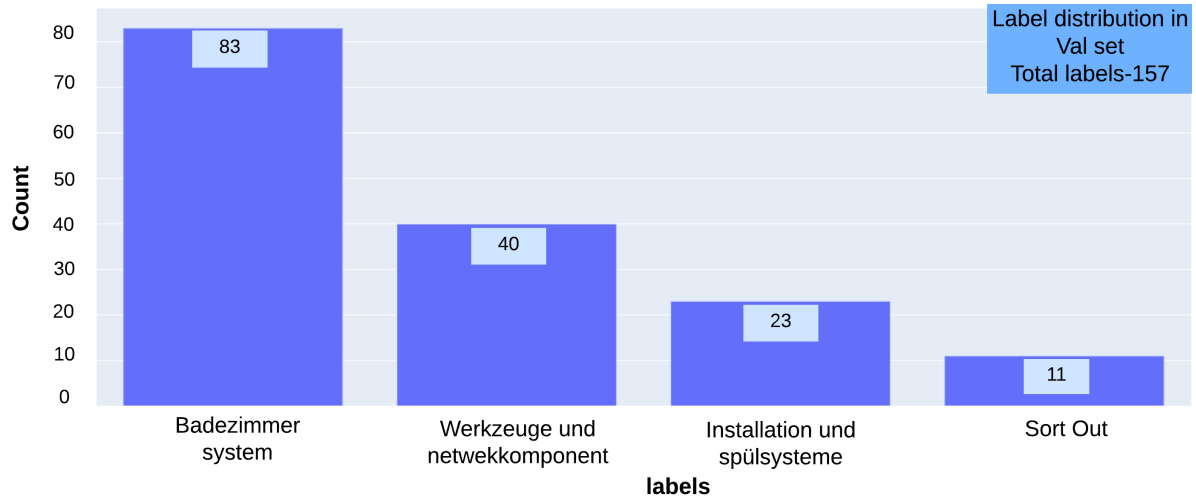

Figure 5.10: Validation Set

Figure 5.11: Split of labeled Data

5.2.2 Data preparation (‘data_prep)̀

The data preparation step, as outlined in section 5.1.2, asynchronously downloads images from Azure Blob storage based on image IDs specified in the CSV file prepared in the previous step. Unlike the approach in MoCo [4] where only a training dataset was required, this step divides the data into separate train, test, and validation sets along with the encoding of their respective labels for evaluation purposes.

5.2.3 Data Training

The training script involves downloading the MoCo V2 model from the Azure Model Store, adding a classification head with the softmax function, and training it on the provided datasets. Several data augmentation techniques were adapted such as color jitter, rotation, flipping, etc. These augmentations are not similar to MoCo V2 transformations as seen previously. However, the goal of augmentations is similar, as it adds variability to the model, making it more robust.

Conventionally, the base model is downloaded from the Pytorch library. Data is then passed through the model and classification head. During training, the loss is computed and quantified. The weights of the neural network are updated by backpropagation using the stochastic gradient descent technique as an optimizer. In the Ideal scenario, trying to minimize the loss iteratively during training. Finally classification report is generated giving detailed insights about the model's performance, including metrics like F1 score, precision, recall, test accuracy, and confusion matrices.

However, for this project, an unsupervised pre-trained model such as MoCo V2 is used as the base model. The model's desired version can be downloaded as a checkpoint file and saved in the output path of the training step. As described in section 5.1.3, only the query encoder without MLP head is attached to the linear classification head from the checkpoint other parameters are neglected. Pytorch does not save the model's hyperparameter in the checkpoint file, as a result, while instantiating the query model class, the same hyperparameters used during pre-training were passed as arguments. The resulting class will act as the base model. The weights of this base model are frozen and only fully connected layers of the classification head are trained with the labeled data.

Successful transfer learning of learned representations relies on knowing the architecture of the base model thoroughly, as the output dimensions or features of the last layer of the base model are the input for the first fully connected layer. The classification head consists of one linear layer, ReLU activation, dropout, and another linear layer in the same order and similar to the one used in the conventional model. It is kept relatively simple and unchanged because this project is a preliminary study and the classes to

identify are small in number.

To summarize, if the training process uses a base model from the Pytorch library it will evaluate the performance of the conventional model and if it uses MoCo V2 model it will evaluate results with pre-training. The evaluation metrics or other aspects of both models remain unchanged.

Chapter 6

Results & Evaluation

This section aims to compare the outcomes of experiments evaluating the effects of pre-training with the Momentum Contrast algorithm on unlabeled data. The first step in this process was independent training of the Momentum Contrast Model. But as we have seen that momentum contrast is a method to learn useful representations and image classification is not possible with this standalone model, a supplementary step was introduced. To address this limitation, a classifier model was developed using pre-trained MoCo representations. To provide a comprehensive evaluation, a conventional machine learning model is also trained with the same parameters, nevertheless without pre-training. This comparative analysis will shed some light on the advantages and efficacy of pre-training, if any.

6.1 Training of Momentum Contrast Model

Since it is an initial study, for the first iteration, the MoCo model was trained using a relatively smaller neural network. As seen in Table 6.1, backbone and backbone_momentum are query and key encoders (ResNet18) while projection_head and projection_head_momentum are their respective MLP head. The total trainable parameters consist of parameters of backbone and projection_head which is approximately 11.5M.

Table 6.1: MoCo Model Components and Parameters.

Name	**Type**	**Params**
backbone	Sequential	11.2 M
projection_head	MoCoProjectionHead	328 K
backbone_momentum	Sequential	11.2 M
projection_head_momentum	MoCoProjectionHead	328 K
criterion	NTXentLoss	0

Trainable Params	11.5 M
Non-trainable Params	11.5 M
Total Params	23.0 M
Time taken to finish training	6 hrs

The important parameters to train the model are listed as follows:

- Input Image Size/Crop Size=100
- Memory Bank size =2048
- Learning rate = 6e-2
- Momentum=0.9
- Weight decay =5e-4
- Number of Images=15000
- Epochs=100
- Batch Size = 128
- GPU – 1 x NVIDIA Tesla T4

The choice of parameter values is contingent on various factors like available computational resources, previous performances on known models, and references from literature reviews. The most important metric as mentioned previously is the behavior of contrastive loss. Figure 6.1 shows that contrastive loss decreases with the increase in epochs.

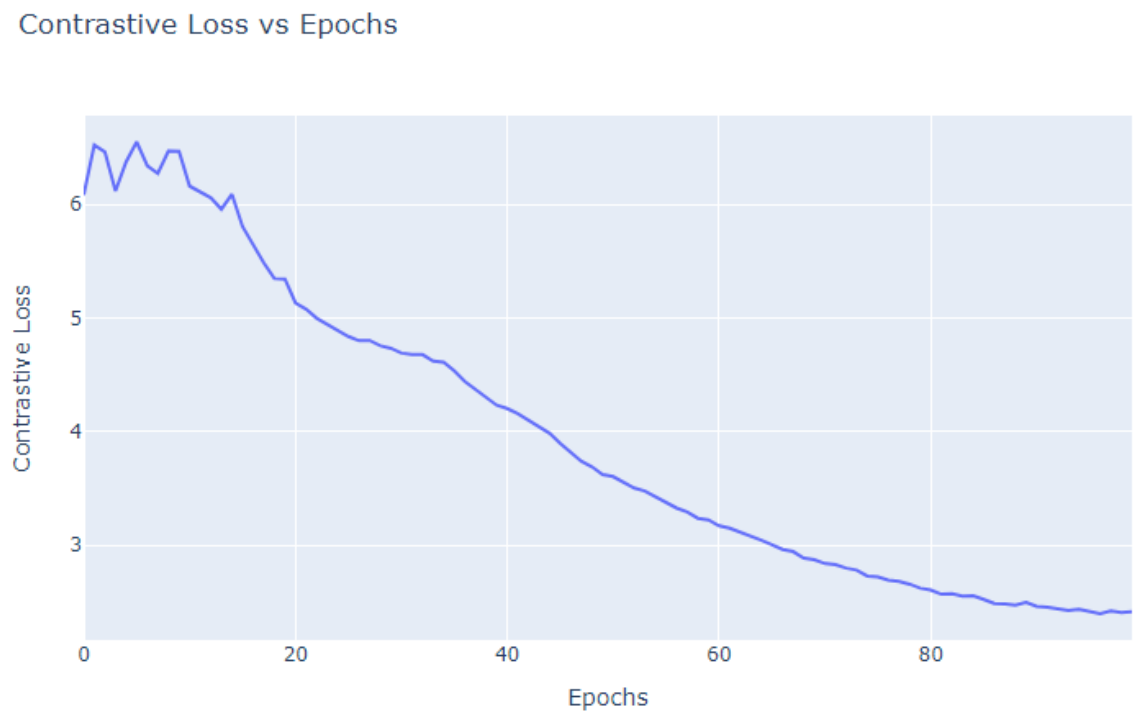

Figure 6.1: Contrastive loss vs Epoch

Decreasing contrastive loss could lead to the following conclusions:

- Model is becoming effective in learning representations that capture relevant information about unlabelled data
- Discriminative features are learned as the positive pairs are getting close and negative pairs are pulled apart. It also shows a sign of convergence to solution which could mean that optimization parameters are working fine.
- The graph does not exhibit large fluctuations after the first few initial epochs, suggesting that the training process is relatively stable

After completing the training process, the model achieved a final contrastive loss of 2.4. The trained model has been saved in the model store and now can be used for inference.

6.2 Classifier Model with Pre-training

In this section, a classifier model is trained with 784 labeled images. The classifier model containsMoCo V2 as the backbone which was trained previously in section 6.1. An appropriate split ratio is defined which splits the labelled data into three sets training, validation, and test as seen in Table 6.2. It is clear from Figure 6.2 that the labeled data for the level-1 hierarchical classifier is highly imbalanced.

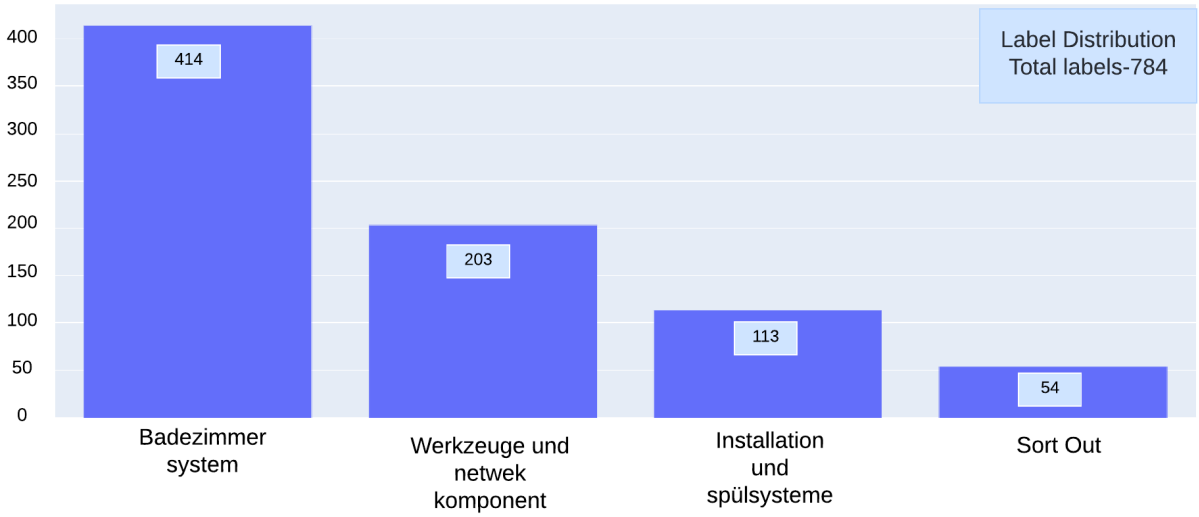

Figure 6.2: Overall label distribution

	Badezimmer Systeme	**Werkzeuge und Netzwerk**	**Sort Out**	**Installations und Spülsysteme**	**Images per dataset**
Available	414	203	54	113	784
Train_split	217	107	28	59	411
Test_split	114	56	15	31	216
Valid_split	83	40	11	23	157

Table 6.2: Class-wise Distribution of Labels

The choice of hyper-parameter was similar to the one defined in pre-training. A slight adjustment to the learning rate was made, reducing it to 0.00005 for training the classification layer. The intuition behind this choice is that a lower learning rate may facilitate a more effective sorting of the classes during the classification process. The results can be observed as follows:

- Training progress:

  The left subplot in Figure 6.3 illustrates a consistent increase in both training and validation accuracy. Simultaneously, the right subplot indicates a decrease in both training and validation loss during the training process. The overall increase in validation accuracy and decrease in validation loss suggests that the model is performing well on unseen data and trying to generalize. Also, no signs of over-fitting or under-fitting are seen.

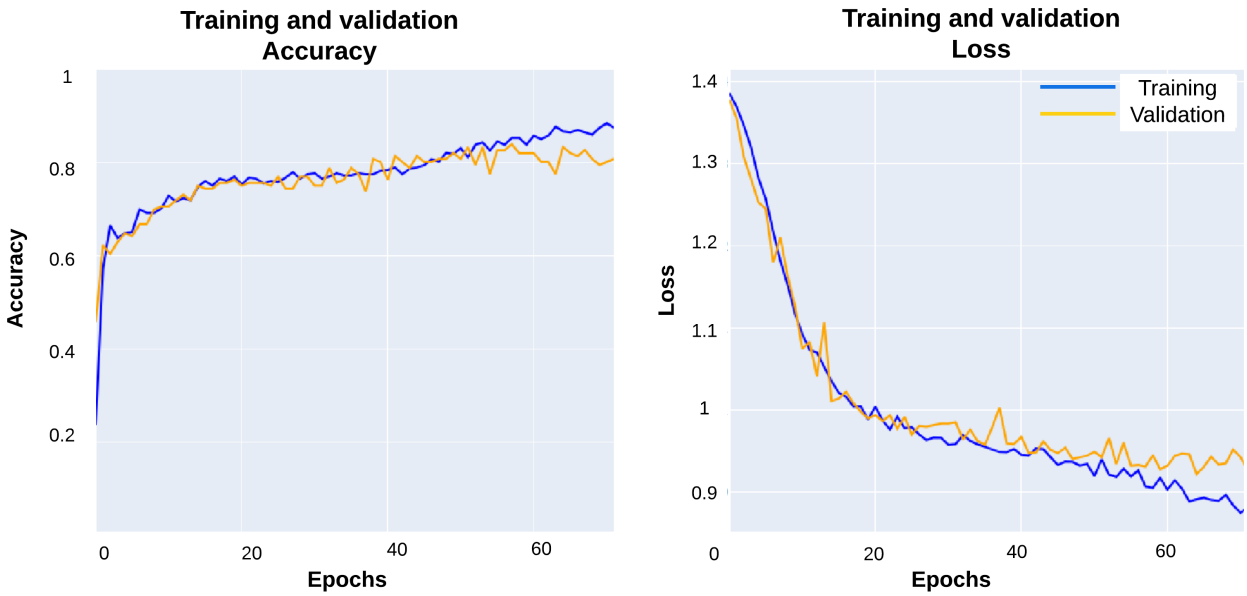

Figure 6.3: Training plots with pre-training
val_loss=0.92, val_accu=0.80

- Confusion Matrix:

  A confusion matrix for the test set was generated, as depicted in the Figure 6.4. While the model demonstrated significant success in generalizing over the unseen data; it can be improved further. A clear observation can be made that the model was able to identify classes such as "badezimmersysteme," "installations und spülsystem," and "werkzeuge und netzwerkkomponenten." However, it was not able to identify a single instance of the "sort-out" class correctly. This behavior can be attributed, at least somewhat, to the imbalance in the dataset. It also shows that the model didn't generalize to all classes.

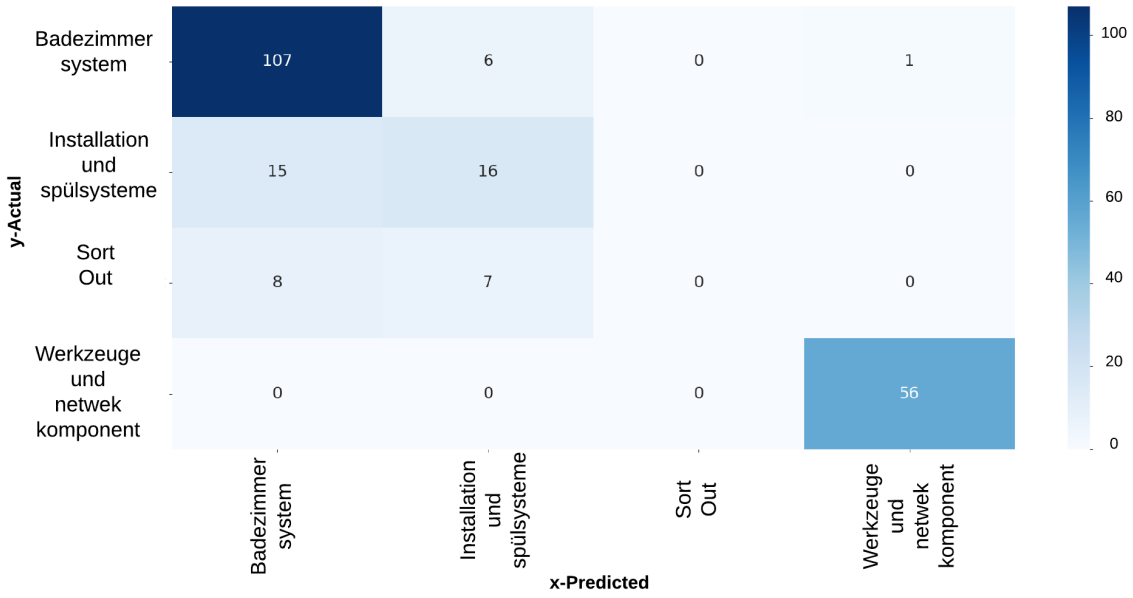

Figure 6.4: Confusion Matrix with pre-training

- Accuracy Metrics:

  As discussed in section 2.2.3, the model's performance is evaluated using metrics such as accuracy, precision, recall, and F1 score. The classification report summarized in Table (6.3) shows that the model is highly effective in predicting the correct instances of the "badezimmersystememe" (88%), and performs moderately well on "installations and spülsystem"(53%). It performed poorly in identifying the "sort out"(0%) samples. While the test accuracy is reported at 82%, it should be noted that this may not be a reliable measure due to the high imbalance in the dataset. Therefore, greater emphasis should be placed on the F1 score(72%) [29].

Table 6.3: Classification Report of Test Set with pre-training

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Badezimmersysteme	0.82	0.94	0.88	114
Installations- und Spülsysteme	0.55	0.52	0.53	31
Sort Out	0.00	0.00	0.00	15
Werkzeuge und Netzwerkkomponenten	0.98	1.00	0.99	56
Overall	0.72	0.73	**0.72**	216

Test accuracy = 0.82

6.2.1 Subapproach-Oversampling

To address the challenge of class imbalance, an oversampling strategy was implemented during the 'Connect_Db' step. Oversampling is a technique in machine learning useful for addressing the challenge of class imbalance. The function counts the instances of the majority class and converts instances of other classes, synthetically, to an equivalent count. The method helps in preventing bias towards the majority class ('Badezimmer-systeme'), allowing the model to better learn from all classes including the minority class ('Sort Out'). Therefore, now we have the same amount of examples for each class for training. It is important to note that samples for the test, and validation set remain the same as shown in the Table 6.2.

- Training progress:

  The introduction of oversampling did not show a significant difference in validation accuracy and loss. While the gap between the training and validation curves is slightly wider compared to the previous scenario seen in Figure 6.3, there is no evidence of significant overfitting or underfitting of the data.

Figure 6.5: Training plots with Oversampling
val_loss=0.95, val_accu=0.78

- Confusion Matrix:

  The confusion matrix shows improvement in classification compared to the previous scenario observed in Figure 6.4. The balancing of classes with the oversampling strategy proved pivotal, enabling the model to successfully classify instances of the "sort-out" class seen from Figure 6.6. This shows that the model was finally able to achieve a certain degree of generalization in all classes.

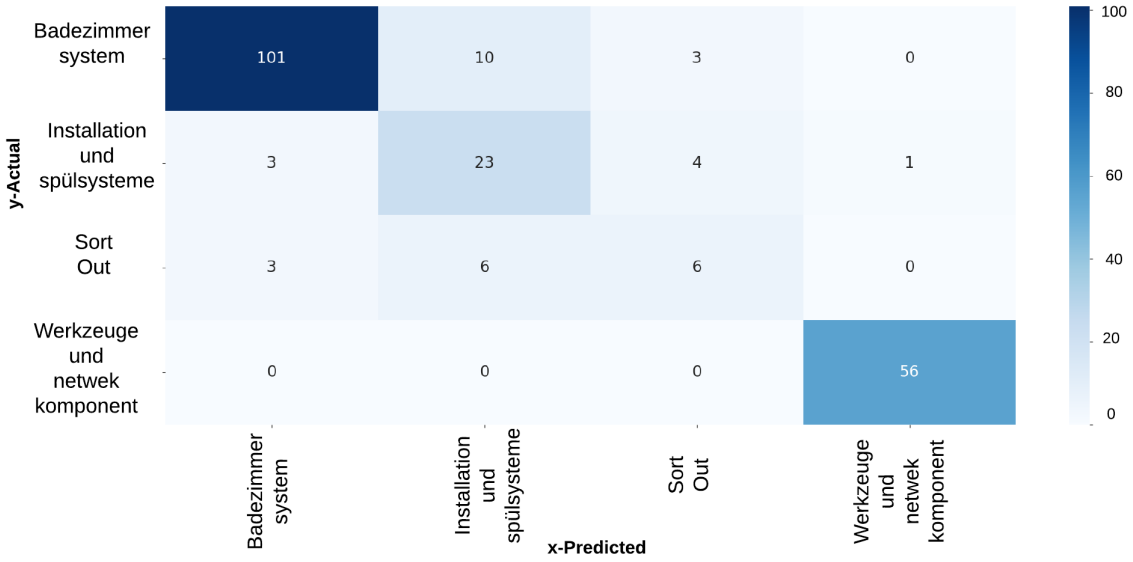

Figure 6.6: Confusion matrix with oversampling

- Accuracy metrics:

  The final classification report of all classes is shown in Table 6.4. The accuracy metrics exhibit performance improvement in all classes, enhancing overall model performance, including the test accuracy and F-1 score.

Table 6.4: Classification Report of Test Set with oversampling

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Badezimmersysteme	0.94	0.89	0.91	114
Installations- und Spülsysteme	0.59	0.74	0.66	31
Sort Out	0.46	0.40	0.43	15
Werkzeuge und Netzwerkkomponenten	0.98	1.00	0.99	56
Overall	0.74	0.75	**0.74**	216

Test accuracy = 0.86

6.3 Conventional machine learning model without pre-training:

In this experiment, a conventional machine learning model is employed for training. The parameters are kept similar to the pre-trained model. The main distinction lies in the absence of pre-training. The base model is adopted as ResNet18 and its weights are frozen, with only the classification head being trained. The classification head is identical to the classifier model in section 5.2. The following results were observed:

- Training progress:

  The training and validation curves seen in the left subplot of Figure 6.7 are stagnant and it would be safe to conclude not much learning is taking place. The model lacks data generalization, and its performance on unseen data might be sub-optimal.

Figure 6.7: Training plots without pre-training

- Confusion Matrix:

  The confusion matrix exhibits high bias towards a specific class, namely “Badezimmersysteme” as seen in Figure 6.8. The model struggled to learn distinguishing features during training, for classifying ”installations-und spülsysteme” and ”sort-out”, underscoring its limitations in achieving robust generalization and performance.

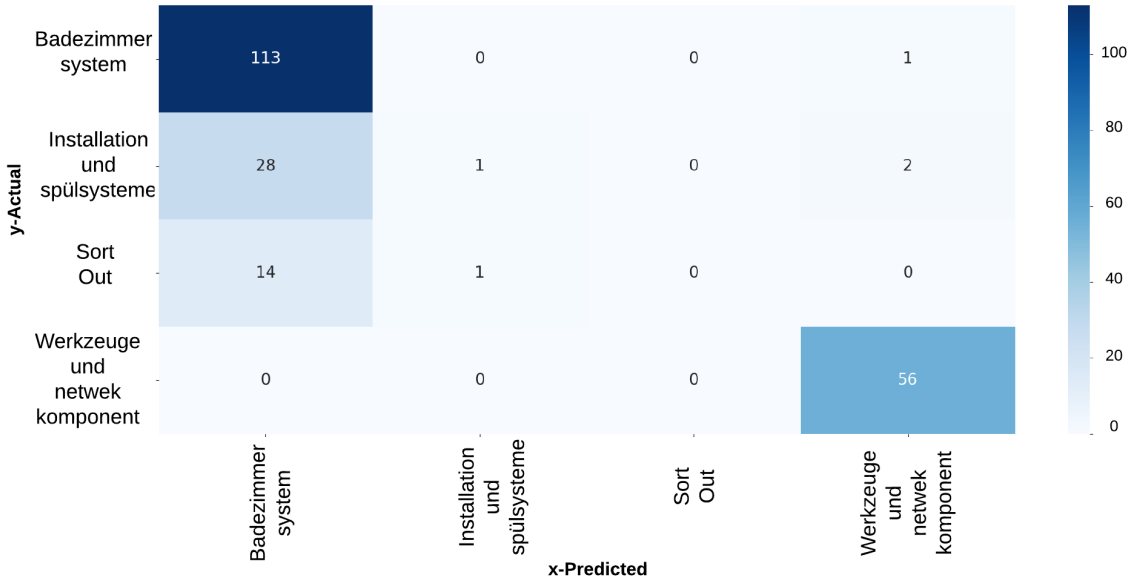

Figure 6.8: Confusion matrix without pre-training

- Classification report on the test set

  The metrics revealed in the table indicate that the model faces challenges in performing well across all classes. It classified most instances as "badezimmer system". The test accuracy and F-1 score are also not promising when compared with pre-training.

Table 6.5: Classification Report of Test Set without pretraining

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Badezimmersysteme	0.73	0.99	0.84	114
Installations- und Spülsysteme	0.50	0.03	0.06	31
Sort Out	0.00	0.00	0.00	15
Werkzeuge und Netzwerkkomponenten	0.95	1.00	0.97	56
Overall	0.54	0.50	**0.46**	216

Test accuracy = 0.78

The results of all three experiments can be summarized with the help of Table 6.6 for comparison and analysis.

Table 6.6: Performance Metrics Comparison with base model as Resnet18

Method	**Precision**	**Recall**	**F-1 Score**	**Accuracy**
Self-supervised	0.72	0.73	0.72	0.82
Self-supervised with oversampling	0.74	0.75	0.74	0.86
Supervised Learning	0.54	0.50	0.46	0.78

Chapter 7

Conclusion, Limitations & Future Work

In conclusion, this thesis has successfully demonstrated the effectiveness of MoCo in image classification. Through experimental analysis presented in the section 6, the study confirms that MoCo as a self-superived learning approach, can learn efficiently learn meaningful representations from unlabeled data and can perform better than the traditional supervised learning method. It outperforms traditional supervised methods in terms of enhanced generalization, F1 score, accuracy, precision, and recall as summarized in Table 6.6.

7.1 Limitations

During the training of MoCo, certain limitations were encountered, such as constraints in GPU capabilities. The current study involved pre-training of 15000 unlabeled image data which took 6 hours to complete. Typically self-supervised methods such as MoCo is trained on a much larger dataset, often in the order of a few hundred thousand, which will require several days to finish with current computational capabilities. Additionally, key parameters like memory bank, image size, and batch size deviated significantly from the recommendations in the MoCo V2 [33] research paper, to prevent GPU memory overflow in Azure. The imbalance in labeled data per category was evident, with some classes having very few instances. This imbalance could significantly impact performance.

7.2 Future work

In the future, enhanced GPU capabilities and better quality of labeled data can aid in performance. Another potential approach could be to adopt data parallel processing techniques. By scaling up the training script with the **Horovod** package and distributing batches across a cluster of nodes simultaneously, computing time could be substantially reduced. For example, if training 300,000 images on a single node of a GPU cluster takes 10 days, parallelization could potentially complete the computation in 2 days by utilizing 5 nodes, with the cost remaining constant and dependent on the number of nodes employed.

With the successful completion of the preliminary phase, experiments can be conducted using a higher volume of unlabeled data to enhance overall performance. Exploring the fine-tuning of the classification head or MoCo model and incorporating batch normalization might prove beneficial. Building on findings from SimCLR [34], which indicate that wider networks lead to improved contrastive loss, it is suggested to use broader networks as backbones for various experiments.

Looking ahead, an immediate future step for this study involves integrating the MoCo pipeline into hierarchical classifiers and conducting inferences to assess performance at different levels of the hierarchy.

Bibliography

[1] Sarker, I. H. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2:1–20, 2021. ISSN 2661-8907. doi: 10.1007/s42979-021-00815-1. URL https://link.springer.com/article/10.1007/s42979-021-00815-1.

[2] Jaiswal, A., Babu, A. R., Zadeh, M. Z., Banerjee, D., and Makedon, F. A survey on contrastive self-supervised learning. *Technologies*, 9:2, 2020. ISSN 2227-7080. doi: 10.3390/technologies9010002. URL https://www.mdpi.com/2227-7080/9/1/2.

[3] Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4): 611–629, August 2018. ISSN 1869-4101. doi: 10.1007/s13244-018-0639-9. URL https://doi.org/10.1007/s13244-018-0639-9.

[4] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning, 2020. URL http://arxiv.org/abs/1911.05722. Comment: CVPR 2020 camera-ready. Code: https://github.com/facebookresearch/moco.

[5] for developers google. Rules of machine learning:. URL https://developers.google.com/machine-learning/guides/rules-of-ml.

[6] Artificial intelligence - symbolic, connectionist, and physical symbol system hypotheses — britannica, 2023. URL https://www.britannica.com/technology/artificial-intelligence.

[7] Oleszak, M. Self-supervised learning in computer vision, 2023. URL https://towardsdatascience.com/self-supervised-learning-in-computer-vision-fd43719b1625.

[8] Intelligence, B. A. Artificial intelligence (ai), 2023. URL https://www.britannica.com/technology/artificial-intelligence.

[9] Muggleton, S. Alan turing and the development of artificial intelligence. *AI Communications*, 27:3–10, 2014. ISSN 09217126. doi: 10.3233/AIC-130579. URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/AIC-130579.

[10] Brownlee, J. A gentle introduction to the gradient boosting algorithm for machine learning, 2016. URL https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning.

[11] Mitchell, T. M. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[12] Sarker, I. H. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2:1–21, 2021. ISSN 2661-8907. doi: 10.1007/s42979-021-00592-x. URL https://link.springer.com/article/10.1007/s42979-021-00592-x.

[13] Liu, Q. and Wu, Y. Supervised learning. 2012. doi: 10.1007/978-1-4419-1428-6_451. URL https://www.researchgate.net/profile/Qiong-Liu-12/publication/229031588_Supervised_Learning/links/551c22380cf2909047ba23f5/Supervised-Learning.pdf.

[14] Mahendra, S. What is supervised learning?, 2022. URL https://www.aiplusinfo.com/blog/supervised-learning.

[15] Dridi, S. Unsupervised learning - a systematic literature review, 2022. URL https://osf.io/kpqr6.

[16] Theissler, A., Pérez-Velázquez, J., Kettelgerdes, M., and Elger, G. Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry. *Reliability Engineering System Safety*, 215:107864, 2021. ISSN 0951-8320. doi: https://doi.org/10.1016/j.ress.2021.107864. URL https://www.sciencedirect.com/science/article/pii/S0951832021003835.

[17] Zhu, X. J. Semi-supervised learning literature survey, 2005. URL https://minds.wisconsin.edu/handle/1793/60444.

[18] Barto, A. G. *Chapter 2 - Reinforcement Learning*, pages 7–30. Academic Press, 1997. ISBN 978-0-12-526430-3. URL https://www.sciencedirect.com/science/article/pii/B9780125264303500039.

[19] Garain, A., Basu, A., Giampaolo, F., Velasquez, J. D., and Sarkar, R. Detection of covid-19 from ct scan images: A spiking neural network-based approach. *Neural Computing and Applications*, 33:12591–12604, 2021. ISSN 0941-0643, 1433-3058. doi: 10.1007/s00521-021-05910-1. URL https://link.springer.com/10.1007/s00521-021-05910-1.

[20] Abas, A. R., Elhenawy, I., Zidan, M., and Othman, M. Bert-cnn: A deep learning model for detecting emotions from text. *Computers, Materials Continua*, 71:2943–2961, 2022. ISSN 1546-2226. doi: 10.32604/cmc.2022.021671. URL https://www.techscience.com/cmc/v71n2/45793.

[21] Wang, X., Zhang, R., Shen, C., and Kong, T. Densecl: A simple framework for self-supervised dense visual pre-training. *Visual Informatics*, 7:30–40, 2023. ISSN 2468502X. doi: 10.1016/j.visinf.2022.09.003. URL https://linkinghub.elsevier.com/retrieve/pii/S2468502X22000936.

[22] Sydenham, P. H. and Thorn, R. *Handbook of measuring system design*. Wiley, 2005. ISBN 978-0-470-02143-9.

[23] Jain, P. Transfer learning and fine tuning of neural networks, 2021. URL https://www.indusmic.com/post/transfer-learning-and-fine-tuning-of-neural-networks.

[24] Financial, S. Mse - mean squared error, 2017. URL https://support.numxl.com/hc/en-us/articles/115001223423-MSE-Mean-Squared-Error.

[25] Bengio, Y., LeCun, Y., and Vazquez, M. Scaling learning algorithms towards ai.

[26] roopam. Convolutional Neural Networks (CNN) Simplified (Part 4), March 2019. URL https://ucanalytics.com/blogs/convolutional-neural-networks-cnn-simplified-part-4/.

[27] Katole, A. L., Yellapragada, K. P., Bedi, A. K., Kalra, S. S., and Chaitanya, M. S. Hierarchical deep learning architecture for 10k objects classification. pages 77–93. Academy Industry Research Collaboration Center (AIRCC), 2015. ISBN 978-1-921987-42-7. doi: 10.5121/csit.2015.51408. URL http://www.airccj.org/CSCP/vol5/csit54508.pdf.

[28] Santra, A. and Christy, J. Genetic algorithm and confusion matrix for document clustering. *International Journal of Computer Science Issues*, 9, 01 2012.

[29] Gaertner, M., Theissler, A., and Fernandes, M. Detecting potential subscribers on twitch: A text mining approach with xgboost - discovery challenge chat: Coolstorybob. 09 2020.

[30] Weiss, K., Khoshgoftaar, T. M., and Wang, D. A survey of transfer learning. *Journal of Big Data*, 3:9, 2016. ISSN 2196-1115. doi: 10.1186/s40537-016-0043-6. URL http://journalofbigdata.springeropen.com/articles/10.1186/s40537-016-0043-6.

[31] 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. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

[32] Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. Unsupervised learning of visual features by contrasting cluster assignments, 2021.

[33] Chen, X., Fan, H., Girshick, R., and He, K. Improved baselines with momentum contrastive learning, 2020.

[34] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations, 2020.

[35] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. .

[36] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. .

[37] Noroozi, M. and Favaro, P. Unsupervised learning of visual representations by solving jigsaw puzzles, 2017. URL http://arxiv.org/abs/1603.09246. Comment: ECCV 2016.

[38] Zhang, R., Isola, P., and Efros, A. A. Colorful image colorization, 2016. URL http://arxiv.org/abs/1603.08511.

[39] Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., and Levine, S. Time-contrastive networks: Self-supervised learning from video, 2018. URL http://arxiv.org/abs/1704.06888.

[40] kundu rohit. The beginner's guide to contrastive learning. URL https://www.v7labs.com/blog/contrastive-learning-guide.

[41] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

[42] seesharprun. Azure Cosmos DB – Unified AI Database, November 2023. URL https://learn.microsoft.com/en-us/azure/cosmos-db/introduction.

[43] seesharprun. Understanding distributed relational databases - Azure Cosmos DB, November 2022. URL https://learn.microsoft.com/en-us/azure/cosmos-db/distributed-relational.

[44] Label Studio Documentation — Overview of Label Studio. URL https://labelstud.io/guide/get_started.

[45] Tsang, S.-H. Review — SimCLR: A Simple Framework for Contrastive Learning of Visual Representations, March 2022.

[46] Ågren, W. The NT-Xent loss upper bound, May 2022. URL http://arxiv.org/abs/2205.03169. arXiv:2205.03169 [cs].

[47] Oord, A. v. d., Li, Y., and Vinyals, O. Representation Learning with Contrastive Predictive Coding, January 2019. URL http://arxiv.org/abs/1807.03748. arXiv:1807.03748 [cs, stat].