The Australian National University
2600 ACT | Canberra | Australia

Australian National University

School of Computing

College of Engineering, Computing and Cybernetics (CECC)

A study of Cross Batch Memory in Deep Metric Learning

— 12 pt research project (S1/S2 2024)

A report submitted for the course
COMP3770, Research and Development Project

By:
Felix O'Brien

Supervisor:
Stephen Gould

October 2024

Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or Wattle site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

October, Felix O'Brien

Acknowledgements

I'm extremely grateful to my supervisor, Professor Stephen Gould, for his guidance and support throughout the project. From initial discussions on projects to advice on the final stages of writing, his expertise has been invaluable. I would also like to thank Thalaiyasingam Ajanthan for his conversations and troubleshooting advice, particularly in addressing my difficulty recreating the cross-batch normalisation results.

I would also like to thank my mother, for her support of me throughout my studies, particularly for providing me with the resources to train many of the models in this paper. I would like to think Dad would have enjoyed reading this.

Abstract

Deep metric learning (DML) has garnered significant attention for its ability to learn a discriminative embedding space, which applies to a wide range of tasks. From retrieval to classification, DML has proven to be a powerful tool for understanding the underlying structure of data. However, the computational cost of DML, particularly due to the requirements of reference sets by contrastive loss functions, can be a significant impediment. To address this, Cross-Batch Memory (XBM) has been proposed as a mechanism to reduce the computational burden by storing and reusing embeddings from previous mini-batches. In this way, larger reference sets can be used without the need to recompute embeddings. Due to its well-established benefits, XBM has seen various modifications in an attempt to improve its performance further. The work in this thesis falls within this line of research, investigating the efficacy of XBM and its variants. Our research introduces three novel modifications: (1) Per-Class normalisation, which performs embedding normalisation independently for each class, better capturing the nuanced distribution of the embedding space; (2) Super-Class Normalisation, which leverages the statistics of superclasses to normalise embeddings of subclasses; and (3) Delayed Normalisation, which postpones the normalisation of the memory bank until the model has reached a certain level of convergence, potentially preventing premature convergence to suboptimal solutions. Through comprehensive experiments on benchmark datasets, this thesis examines the effectiveness of our proposed modifications in enhancing the efficiency and performance of DML models. We also re-evaluate the efficacy of XBM and its existing variants, providing a comprehensive study of the current state of the art in DML research.

Table of Contents

Chapter 1

Introduction

Deep Metric Learning (DML) has emerged as a powerful technique for learning representations that capture the underlying similarity relationships between data points. Using deep neural networks, DML approaches map data of various modalities, such as images, audio, and text, into high-dimensional embedding spaces (Mohan et al., 2023a). By mapping data into a discriminative embedding space, DML facilitates various downstream applications, including image retrieval, clustering, and few-shot learning (Mohan et al., 2023a) (Snell et al., 2017). The effectiveness of DML hinges on its ability to learn an embedding space that reflects the semantic similarity between data points, bringing similar items closer together in the embedding space while pushing dissimilar items farther apart.

The increasing availability of large-scale datasets and the advancements in deep learning architectures have fuelled the progress of DML, leading to significant improvements in performance across various tasks. However, the computational cost associated with training DML models, particularly due to the use of contrastive loss functions that require comparisons between numerous pairs, triplets, or larger groups of data points, remains a major challenge. This computational burden can hinder the scalability of DML to massive datasets and limit its applicability in resource-constrained environments.

To address this challenge, Cross-Batch Memory (XBM) (Wang et al., 2019) has been proposed as an efficient approach to reduce the computational cost of DML. XBM leverages the observation that embeddings for a given data point tend to evolve gradually during training, allowing the reuse of embeddings from previous mini-batches to approximate the current state of the embedding space. By caching and reusing these embeddings in an external memory bank, XBM eliminates the need for recomputing embeddings for the entire dataset at each training iteration, thereby significantly decreasing the computational overhead and allowing for much more significant learning.

While XBM offers a promising solution for efficient DML, it does not explicitly account

for the dynamic nature of the embedding space during training. As the model learns and the embedding space evolves, the stored embeddings in the memory bank may become outdated, leading to suboptimal performance. To mitigate this issue, normalised Cross-Batch Memory (XBN) (Ajanthan et al., 2023) was introduced, which normalises the stored embeddings based on the statistics of the most recent mini-batch. However, XBN relies on global normalisation, potentially overlooking the nuanced and class-specific dynamics of representational drift.

This thesis delves into the intricacies of XBM and proposes novel modifications to enhance its efficiency and adaptability in DML further. Specifically, we examine three novel modifications to the XBM framework

- **Per-Class Normalisation** A more fine-grained normalisation scheme is proposed that operates independently for each class, capturing the class-specific dynamics of representational drift.
- **Super-Class Normalisation** For scenarios with limited samples per class, a normalisation technique is introduced that leverages the statistics of broader superclasses to normalise the embeddings of their constituent subclasses.
- **Delayed Normalisation** A strategy is proposed to postpone the normalisation of the memory bank until the model has reached a certain level of convergence, potentially preventing premature convergence to suboptimal solutions.

The work in this thesis also re-evaluate the efficacy of XBM and its existing variants, providing a comprehensive study of XBM as a class of approach. Through comprehensive experiments on benchmark datasets, this thesis demonstrates the effectiveness of the above modifications in improving the performance and efficiency of DML models. The findings contribute to the understanding of XBM and provide valuable insights for designing more adaptive and robust DML techniques.

Chapter 2

Background

It is first necessary to outline the relevant mathematics and theory which lies behind the research, particularly focusing on Metric Learning and Deep Learning.

2.1 Metric Spaces

Central to the research is the metric space, the space to which the dataset is mapped. A metric space gives us a notion of distance between two points, which allows for discrimination between samples.

Definition 1. A metric space is a set M equipped with a real-valued function $D(a, b)$ defined for all $a, b \in M$ which satisfies the following properties:

1. Positivity: $D(a, b) \geq 0$ and $D(a, b) = 0 \iff a = b$
2. Symmetry: $D(a, b) = D(b, a)$
3. Triangle Inequality: $D(a, b) + D(b, c) \geq D(a, c)$

Kaplansky (1977)

Within metric learning, the distance function measures the similarity between two samples, where similar samples are closer in the embedding space than dissimilar samples. The question then arises as to how to define the distance function. Perhaps the most common choice is Euclidean distance,

$$D(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{2.1}$$

A more general notion of distance is the Mahalanobis distance, which scales the distance across the dimensions of the embedding space by a positive definite matrix M. In this way, the metric considers the data distribution when calculating the distance between two points.

$$D(a, b) = \sqrt{(a - b)^T M (a - b)} \tag{2.2}$$

When the matrix M is an identity matrix, the Mahalanobis distance reduces to the Euclidean distance. While there are applications of the Mahalanobis distance in metric learning, it can only learn a linear transformation of the data. This is a significant limitation when considering any non-linear relationships that may exist in the data. As such, the Mahalanobis distance is not used in this research.

Another important notion is the vector norm, which generalises length. The norm of a vector x is denoted as $||x||$, and the commonly used norm is the Euclidean norm, defined as the square root of the sum of the squares of the vector's components.

$$||x|| = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{2.3}$$

The norm proves to be useful for normalising vectors, which is a common practice in metric learning. Normalisation is simply scaling the vector by division of the norm of the vector such that it has a unit norm. Vectors with a unit norm lie on the unit sphere or, in the case of multiple dimensions, the unit hypersphere. The benefit of normalisation is that it removes the magnitude of the vector, reducing the comparison of vectors to their direction. The means of this comparison introduces a notion of similarity, but not a metric, in cosine similarity.

Definition 2. The cosine similarity between two vectors a and b is defined as:

$$\cos\theta = \frac{a \cdot b}{||a||||b||} \tag{2.4}$$

Continuing from the dot product, where $a \cdot b = ||a||||b|| \cos\theta$, the cosine similarity measures the angle between two vectors. As a result of the above normalisation, the cosine similarity between two vectors can be reduced to the dot product of the two vectors. Intuitively, this produces a value between -1 and 1, where 1 indicates the vectors are in the same direction, -1 indicates they are in opposite directions, and 0 is orthogonal. This is the approach used in this research.

2.2 Metric Learning

Metric Learning has been a field of interest for several decades and is the precursor to modern deep metric learning approaches, which is explored in this work. Inherent in

the name is the notion of learning a metric for given data, that is, learning a distance function that can be used to discriminate between samples. At the lowest level, one can apply nearest neighbour distance algorithms to data in a metric space for classification. One such approach is K-Nearest Neighbours (k-NN), a classification approach that uses the k nearest samples to an unseen data point to vote on that label's class (Hertz, 2006).

The k-NN algorithm provides a base from which one can apply more advanced techniques. As a result of the reliance on the distance between a point and its neighbours, improving the metric used when considering the nearest neighbour sampling can improve the performance of the k-NN algorithm. Such applications include using local class densities and their gradients to improve the metric or using a Mahalanobis metric to learn the distance metric (Hertz, 2006). Mahalanobis metric learning algorithms use a weight matrix to learn the distance metric and may be adjusted via gradient descent, ascent or other optimisation algorithms (Hertz, 2006). It is important to recognise that, unlike Deep Learning applications, these approaches aim to learn the distance metric rather than the embedding space itself.

An approach that is closer to later Deep Learning approaches and those explored within this project is that of Zhang et al. (2003), which uses a similarity that considers class labels to learn an embedding space. A regression model is used to learn the mapping from the input space to the feature space. The limitation of this approach is again that it is linear and, therefore, cannot learn non-linear relationships in the data. This is a significant limitation when considering the complexity of the data often used in metric learning tasks. Deep Learning approaches are particularly useful for addressing the challenges of non-linear transformation.

2.2.1 Evaluation Metrics

Within the metric learning field, various evaluation metrics can be used to assess the quality of the learned embedding space and, thereby, the quality of the model. Whilst various metrics can be used, the most common are those that are derived from information retrieval tasks. This is because the metric learning task is similar to information retrieval, where the goal is to retrieve the most similar items given a query. In this work, the Recall@K metric, particularly Recall@1 and Recall@10, are used as the primary evaluation metrics. It is important to note that in the metric learning context, Recall@K differs from the traditional definition of Recall. Rather than representing the proportion of relevant items retrieved in the top K items, Recall@K in the metric learning context is 1 if the expected class is in the top K retrieved items and 0 otherwise. This is similar to top-k accuracy. This result is averaged over all queries to produce the final Recall@K score.

Definition 3. Recall@K is the percentage of queries for which the expected class is in

the top K retrieved items.

$$(Metric)Recall@K = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}1 & \text{if expected class is in top K}\\ 0 & \text{otherwise}\end{cases} \tag{2.5}$$

One benefit of the Recall@K, as opposed to more complicated metrics, is that it directly measures the quality of the learned embedding space. That is, it measures the ability of the embedding space to discriminate between samples. If we were more concerned with the ranking of the samples, then the Mean Average Precision (mAP) metric would be more appropriate. However, the Recall@K metric is more appropriate in the context of metric learning as it is one of the most common evaluation metrics used (Musgrave et al., 2020a). This in itself makes it a valuable metric to enable comparison across different approaches.

2.3 Deep Learning

Stepping away from metric learning, Deep Learning has seen an explosion in popularity in the last decade. Built upon the foundations of Neural Networks (NNs), Deep Learning has been applied to a wide range of applications, from image recognition to natural language processing. Their relevance to metric learning comes from their ability to perform complex transformations, such as the transformation from input data to an embedding space guided by a desired objective function. The earliest form of NNs were perceptrons, consisting of cells (neurons) connected from an input in layers to a desired output. The neuron consists of weights that dictate the contribution a given input has on the perceptron's output and an activation function that fired 1 when the sum of all inputs into a neuron exceeded a given threshold and 0 otherwise (Rosenblatt, 1958). Importantly, Rosenblatt (1958) introduced a learning algorithm that allowed the perceptron to adjust its parameters based on the error. Multi-layer perceptrons (MLPs) were introduced to allow for more complex transformations. The backpropagation algorithm's introduction improved neural network learning and provided a standardised training process in the form of gradient descent. This algorithm calculates the gradient of the loss function concerning the network weights. This gradient is then propagated back through the network and used to update the network weights in the opposite direction of the gradient. Rumelhart et al. (1986) popularised the backpropagation algorithm used as the learning algorithm by most NNs today. Backpropagation enables objective functions that describe a desired embedding space or properties of an embedding space to be optimised for metric learning. Importantly, its efficacy depends on the gradients of the loss function, with significant gradients increasing the learning rate of the network. As such, issues arise with objective functions or training contexts where the gradients are small, leading to slow learning or no learning. The reduced gradient problem is a significant issue that cross-batch memory aims to address (Wang et al., 2019). Since then, NNs have evolved significantly, with numerous different architectures and learning algorithms. Most importantly, we use NNs because of their ability to learn complex

non-linear relationships between input and output data; NNs are universal function approximators Schmidhuber (2014) Hornik et al. (1989). As a result, relating to metric learning, NNs can be used to learn a (non-linear) transformation from arbitrary input data to an embedding space. The ability to learn non-linear transformations is the basis of Deep Metric Learning approaches like the one explored in this research.

2.3.1 Convolutional Neural Networks

A subset of NNs are Convolutional Neural Networks (CNNs), particularly well-suited to image data. CNNs use convolutional layers consisting of learned kernels to learn features from images. Fully connected layers often follow this. Convolutional layers significantly reduce model complexity by reducing the dimensionality of the data and assist with the feature extraction of the later layers (O'Shea and Nash, 2015). CNNs have been used in many applications, from image classification (Krizhevsky et al., 2012) to image generation (Goodfellow et al., 2014). The fully connected layers can be adjuster to encode a desired set of features from the information the convolutional layers extract. The generic feature outputs make CNNs well suited to metric learning for image data. In a supervised setting, the final fully connected layer is often paired with a softmax activation function to produce class probabilities. A downside to this approach is that associated objective functions do not respond well to high inter-class and low intra-class variances (Mohan et al., 2023a). This is where metric learning approaches are more effective.

2.3.2 Residual Networks

Of particular relevance to this project is Residual Networks (ResNets), which are a type of CNN that uses residual blocks to learn the residual input data. A residual block consists of a series of convolutional layers with a skip connection that adds the input to the output of the convolutional layers (He et al., 2015). Figure 2.1 shows a visualisation of a residual block.

The residual blocks make Resnets particularly good feature extractors as they both allow more depth in the model and represent more complex relations throughout the model (He et al., 2015). Adding a fully complete layer after the final convolutional layer allows for task-specific learnings, such as mapping to a desired dimension embedding space. Such properties make ResNets ideal for use in metric learning, just as they remain the gold standard architecture for recent scientific publications. The ResNet50 architecture consists of 50 layers and is used as the base model for this project. At its release, ResNet50 achieved state-of-the-art performance in image recognition on the ImageNet dataset Deng et al. (2009), the dataset on which we train the model used in this research. Its architecture is depicted in Figure 2.2.

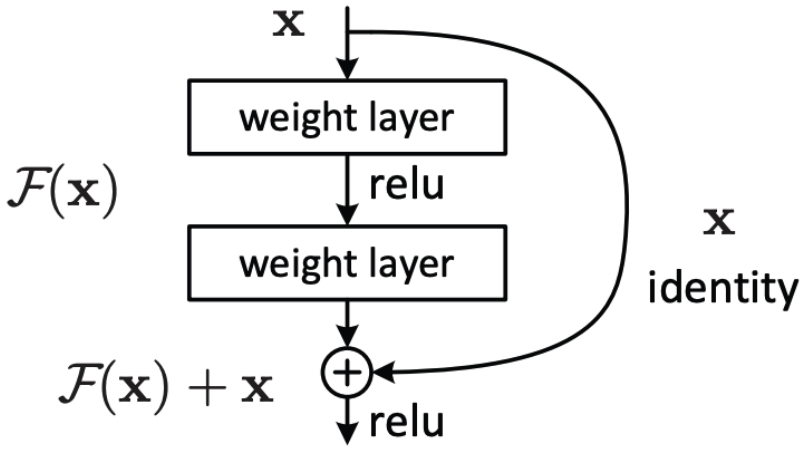

Figure 2.1: Residual Block (He et al., 2015)

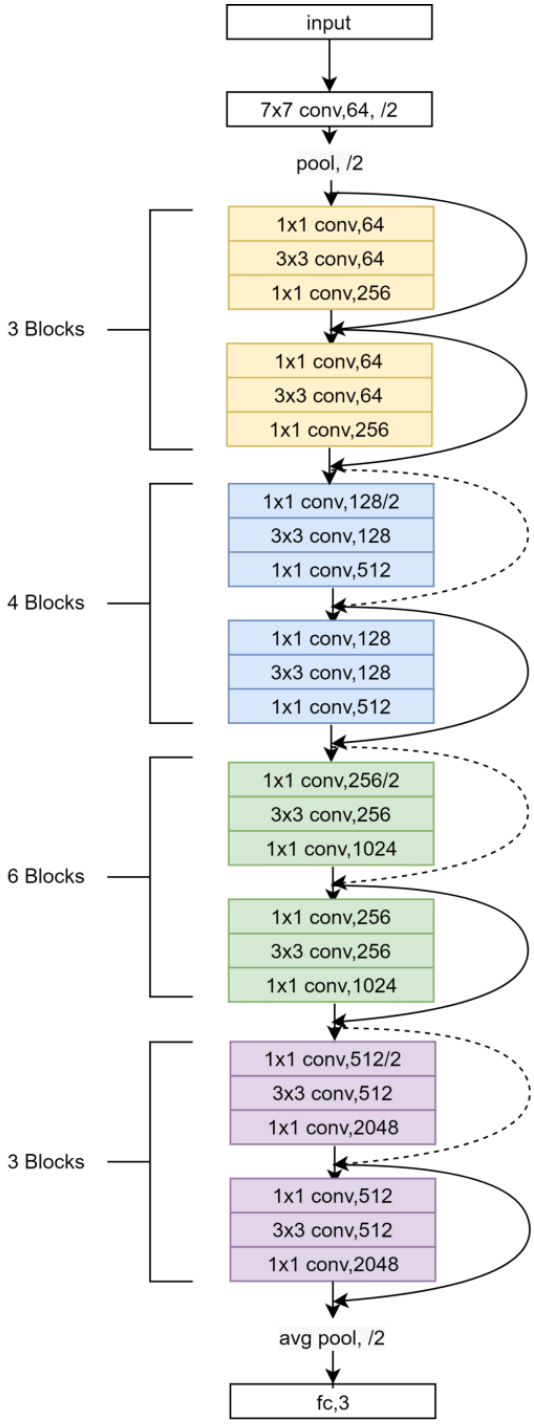

Figure 2.2: ResNet50 Architecture (He et al., 2015)

Chapter 3

Related Work

This chapter provides an overview of the literature on deep metric learning in general and cross-batch memory in particular. Building on the background, various loss functions used in deep metric learning are explored, followed by a discussion of mining and cross-batch memory. Finally, we discuss the existing modifications to cross-batch memory.

3.1 Deep Metric Learning

Deep Metric Learning (DML) is the application of deep neural networks to learn a mapping from input data to an embedding space where the semantic similarity between data points is encoded. This is achieved by training the network to minimise the distance between similar data points and maximise the distance between dissimilar data points.

3.1.1 DML Applications

Typically, DML is applied to scenarios where

1. The data is high dimensional, such as images, audio or text
2. The data is non-linearly separable
3. The data has low inter-class variance and high intra-class variance (Mohan et al., 2023b)

These properties mean that DML is particularly well suited to the computer vision field, where applications from image retrieval to image classification and clustering have been explored.

One such application is face recognition, a task characterised by the often low sample size of each class, high dimensionality of the data and high intra-class variance due to different poses, lighting conditions and facial expressions (Schroff et al., 2015). As

mentioned earlier, DML approaches are often used on top of a CNN backbone because of their strong ability to extract lower dimensional features from high dimensional data, which can then be used as embeddings. This approach is demonstrated by Schroff et al. (2015), who use a deep CNN to extract features for face recognition using a triplet loss to learn the embedding space. One benefit that Schroff et al. (2015) found in pairing DML with a CNN is that it significantly reduces the required dimensionality of the output embedding when compared to other DML backbone combinations.

More generally, few-shot classification problems are well suited to DML approaches because the learning is the embedding space, meaning a model can be trained on a small number of samples and still generalise well to unseen data. This is particularly useful in scenarios where the number of classes is significant but the number of samples per class is small. Prototypical Networks, a DML approach that takes the mean of the embeddings of each few-shot class to create a prototype, is particularly effective in few-shot classification tasks (Snell et al., 2017). Using softmax distances to the prototypes to provide a class probability highlights one method for using DML in classification tasks. The notion of using a prototype to represent a class is similar to the idea of using the class mean as a normalisation statistic in the PC-XBN approach we explore.

3.1.2 Loss Functions

Now that the applications of DML have been outlined, the loss functions by which the model learns the embedding space can be considered more deeply. One must choose an appropriate objective function to create a highly separable embedding space. As outlined in the introduction, most of these approaches require comparing samples and their points in the embedding space. We only consider other task-based loss functions with these properties, such as classification losses, as cross-batch memory best suits contrastive learning approaches. That is not to say we cannot use cross-batch memory in other contexts, but most of the literature focuses on contrastive learning approaches.

The baseline for contrastive learning is the simple contrastive loss, which aims to pull samples of the same class closer together and push samples of different classes further apart (Hadsell et al., 2006). The contrastive loss is defined as follows:

$$L(y, x_1, x_2) = (1 - y)\frac{1}{2}||f(x_1) - f(x_2)||^2 + y\frac{1}{2}\max(0, m - ||f(x_1) - f(x_2)||)^2 \quad (3.1)$$

where y is 1 if the samples are of the same class and 0 if they are of different classes. The constant m is the margin within which samples of the different classes are deemed too close. Contrastive loss is often seen as baseline performance in deep metric learning tasks, and further improvements are often made by increasing its complexity (Musgrave et al., 2020a).

An improvement of a simple contrastive loss is Triplet Loss, using three embeddings in the sample space: the positive sample, an anchor of the same class as the positive sample and a negative sample of a different class. This loss simultaneously considers both the

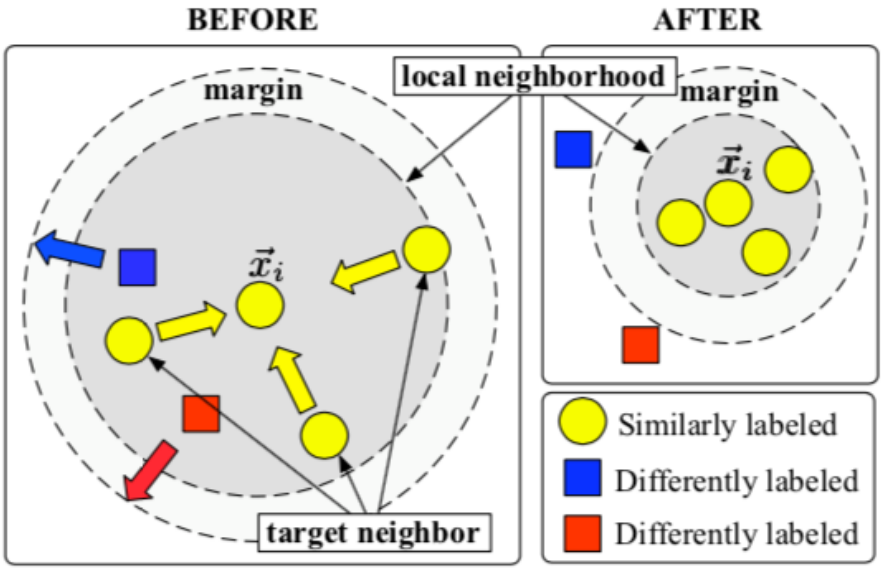

Figure 3.1: Visualisation of Contrastive Loss (Khosla et al., 2020a)

distance between a given sample and its same-class anchor and the distance between the sample and its different-class negative (Hoffer and Ailon, 2014).

$$L(x, x^+, x^-) = ||f(x) - f(x^+)||^2 - ||f(x) - f(x^-)||^2 \tag{3.2}$$

where $f(x)$ is the embedding of sample x, x^+ is the positive sample and x^- is the negative sample.

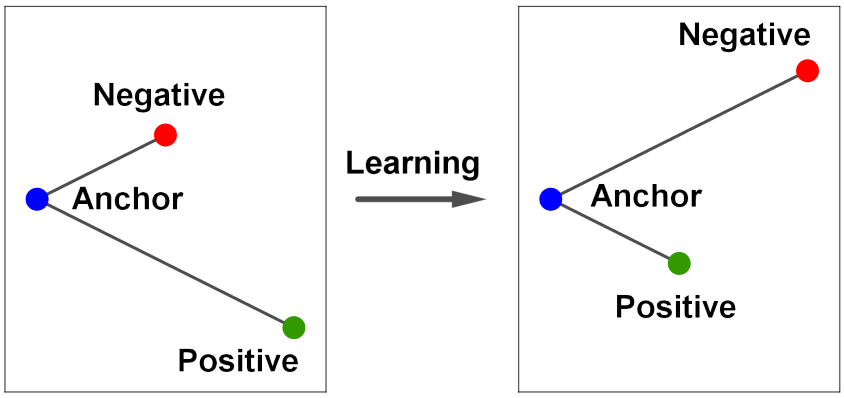

Figure 3.2: Visualisation of Triplet Loss (Hoffer and Ailon, 2014)

In this way, the positive is pulled towards the anchor while the negative is pushed away. A downside of this approach is that samples will not contribute gradients of significant magnitude to the loss function if they are already well positioned. Positive samples close to the anchor and negative samples far from the anchor contribute little to the loss. Instead, there is a reliance on hard positive and hard negative samples, samples of the same class far away from an anchor and samples of a different class close to an anchor.

Building from Triplet Loss, N-Pair loss is considered a positive sample, and N-1 is a negative sample concerning an anchor. In doing so, the N-Pair loss guarantees the positive sample is far from the selected negative class and far from the N-2 other classes (Sohn, 2016). Despite this, Triplet and N-pair losses are limited in considering only one positive for each anchor, while many positives likely exist for a given anchor. Supervised Contrastive Loss contrasts the set of all samples from the same class as positives against the negatives from the remainder of the batch Khosla et al. (2020b).

$$L(x) = -\sum_{x^+} \log \frac{\exp(f(x) \cdot f(x^+)/\tau)}{\sum_{x^- \in X \setminus \{x\}} \exp f(x) \cdot f(x^-)/\tau} \tag{3.3}$$

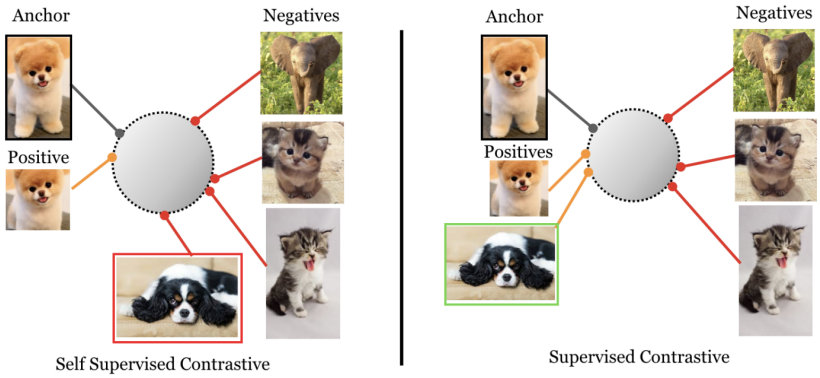

Figure 3.3: Visualisation of SupConLoss (Khosla et al., 2020a)

This approach is beneficial in scenarios where the number of classes is significant, but the number of samples may be small, as it helps to balance out the number of positives when compared to single positive approaches (Khosla et al., 2020b); the use of SupConLoss significantly improves performance on classification losses. As such, SupConLoss is particularly well suited to this research due to the use of industry-standard benchmark datasets In-Shop (Liu et al., 2016), SOP Song et al. (2016), and DeepFashion2 Ge et al. (2019) which have a large number of classes but a small number of samples per class.

One final loss function to consider is the Centre Contrastive loss (CCL) (Cai et al., 2023), which maintains a bank of class centres to compare samples to standard contrastive loss.

We consider this loss function because of its similarity to cross-batch memory, which also holds a bank of embeddings to which it compares samples.

$$\mathcal{L} = \mathcal{L}_{\text{contrastive}} + \lambda\mathcal{L}_{\text{center}} = -\log \frac{e^{s\cdot\left(c_y^T x - m\right) + 2\lambda c_y^T x}}{e^{s\cdot\left(c_y^T x - m\right)} + \sum_{j=1}^{K} e^{s\cdot\left(c_j^T x\right)}} \tag{3.4}$$

where c_y is the class centre, x is the embedding, m is the margin, s is the scaling factor and λ is the weighting factor.

Unlike cross-batch memory, CCL maintains class centres rather than the entire embedding space. In that way, CCL does not provide additional pairs to the miner. Still, the loss considers the distance between the embedding and the class centre and the distance between the embedding and the anchor or negative.

This suggests that encouraging the embeddings to be close to the class centres, in addition to maintaining the contrastive separability of the embedding space, can improve the model's performance. This work's per-class normalised cross-batch memory approach is conceptually similar to CCL; normalising the embeddings to the class implicitly encourages the stored embeddings to be close to the class centre. As such, CCL is an additional motivation for the per-class normalised cross-batch memory approach in this work.

From the above, it is clear that popular loss functions carry many fundamental similarities. Empirical research suggests that more complicated loss functions only slightly increase the practical performance of deep metric learning models when compared to baseline contrastive loss performance (Musgrave et al., 2020a). For this reason and because of the similar nature of approaches, multiple loss functions are not explored in this research. Additionally, most loss functions explored share the same fundamental structure, building from positives and negatives, making the influence of cross-batch memory similar across the loss functions. That said, Supervised Contrastive Loss is employed in this research to replicate and compare our extensions against the setup of Ajanthan et al. (2023).

3.1.3 Mining

As all of the losses explored above require some form of positive or negative classification alongside a possible anchor, selecting such samples is a critical part of the training process. The selection of these samples is referred to as mining. To maximise the efficacy of mining, it is desirable to select samples that are the least ideal and, therefore, contribute most significantly to the loss. This, in turn, maximises the possible gradient contribution to the loss function. In the case of triplet loss, the optimal selection is to select the closest negative sample to the anchor and the furthest positive sample, i.e. $argmax_{x^-}||f(x) - f(x^-)||^2 - argmin_{x^+}||f(x) - f(x^+)||^2$ (Schroff et al., 2015). These samples are deemed hard negatives and hard positives. As Schroff et al. (2015) states, it is infeasible to compute these samples for the entire dataset, with a $\mathbb{O}(N^3)$ complexity

for evaluating all triplets (Harwood et al., 2017). Instead, a common approach is to use semi-hard mining, which selects negative samples further from the anchor than the positive sample. Compared to hard-mining, an advantage of this approach is that choosing the hardest negative samples can lead to a model that collapses to a local minimum (Schroff et al., 2015). Applying semi-hard mining over a subset of samples in each minibatch further reduces the computational complexity of the loss function (Harwood et al., 2017).

Providing a more extensive reference set from which pairs or triplets can be chosen is the fundamental premise of cross-batch memory. The introduction of a memory bank allows for the selection of more informative hard pairs, as the miner can select from the entire memory bank rather than just the current minibatch, providing a more stable training process and more significant learning as a result of the increased gradient contribution to the loss function (Wang et al., 2019) (Harwood et al., 2017).

3.1.4 Cross-Batch Memory

With the complexity of contrastive losses clearly outlined, an alternative approach to improving training efficiency for DML applications is to reduce the scale of data throughput through the model when calculating the loss. Wang et al. (2019) proposed the introduction of an external queue-like memory bank into which the most recently computed embeddings of a given training minibatch are inserted. Built over multiple mini-batches, the XBM provides the reference set for the contrastive loss of the current minibatch, as shown in Figure 3.4.

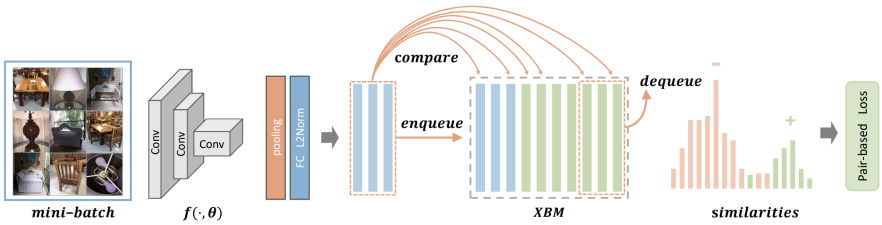

Figure 3.4: XBM Method (Wang et al., 2019)

The introduction of XBM is based on evidence that beyond the early stages of model training, the embeddings produced by an input sample are unlikely to change significantly over time, a phenomenon termed "slow-drift" (Wang et al., 2019). As such, current embeddings can be approximated with stale embeddings to a valuable degree for inclusion as additional reference embeddings in the loss function.

$$D(x, k, \Delta k) = ||z^k - z^{k-\Delta k}||_2 \tag{3.5}$$

Feature drift as described in Wang et al. (2019), where z^k is the embedding of sample x at time k and Δk is the time difference between the embeddings.

The benefit of this process is it avoids the recomputation of embeddings for the entire dataset or significant portions of it at each minibatch. A practical benefit lies in the increase of informative hard pairs from which a miner can select, as the miner can now select from the entire memory bank rather than just the current minibatch. This results in a more stable training process and significant learning due to the increased gradient contribution to the loss function (Wang et al., 2019) (Harwood et al., 2017). Additionally, it avoids the issue of GPU memory limitations that arise when increasing the batch size to increase the number of semi-hard pairs. XBM's success is demonstrated in the byt Wang et al. (2019), where the XBM approach significantly outperforms non-XBM approaches. It is worth noting that beyond the original paper, the success of XBM could have been better, with further released attempts instead demonstrating slightly reduced performance (Ajanthan et al., 2023).

In a similar vein is the approach of Cross-Batch Negative Sampling for Training Two-Tower Recommenders, which uses an identical queue-like memory bank for caching previous embeddings as a means to increase the number of negative samples available for training Wang et al. (2021). Additionally, Wang et al. (2021) recognise the same premise of slow-drift in the embeddings post the initial training phase, which justifies the caching of old and stale embeddings. A similar positive result is found in the performance of the two-tower recommender system when compared to the traditional approach.

3.1.5 Normalised Cross-Batch Memory

As mentioned above, relying on the slow drift of features to ensure the closeness of the approximations in XBM has been shown to have positive results; that is not to say adjustments cannot be made to XBM approaches to better track feature learning. One proposed solution is the normalisation of embeddings stored in XBM to the first two moments (mean and standard deviation) of the recently processed minibatch (XBN) embeddings.

$$\hat{z} = \frac{z - \mathbb{E}[R]}{\sigma[R]} \sigma[B] + \mathbb{E}[B] \tag{3.6}$$

Such a cross-batch memory with a normalisation approach shown in Eq. 3.6 has been demonstrated to have increased performance when compared to the traditional approach Ajanthan et al. (2023). This is attributable to the fact that stored embeddings are transformed through the applied normalisation to better represent recent changes in the model after the backward pass of the recent mini-batch(es). This approach adds very little overhead to the XBM approach, with two lines of code implementing the normalisation (Ajanthan et al., 2023).

One concern outlined in the original paper is the instability of the mean and variance of the embeddings. The recent minibatch can only provide a noisy estimate for the learning of the model, particularly under the small batch size of 64 samples used by Ajanthan

et al. (2023). This is a significant issue as the mean and variance are used to normalise the embeddings in the memory bank. To address this, the authors propose introducing a Kalman filter to better estimate the first two moments of the recent minibatch, given the reduced size. A Kalman filter (Kalman, 1960) aims to create estimates of statistics from noisy samples such as those taken from successive mini-batches. However, introducing a Kalman filter was only shown to slightly improve the performance of the model compared to standard XBN Ajanthan et al. (2023).

We find one slight technical oversight in their method when reviewing the experimental setup used in the XBN approach. Throughout the experiments, the authors use a `PairMarginMiner` with a positive margin of 0.2 and a negative margin of 0.8. These are the default parameters for the pytorch-metric-learning library. Interestingly, these parameters, as well as the `pytorch-metric-learning library` document, suggest that a default the miner employs the default Euclidean distance rather than the CosineSimilarity that is used elsewhere in the XBN approach. Using Euclidean distance in the miner would not be a problem if all the embeddings lay on the unit hypersphere. Still, the normalisation approach employed by Ajanthan et al. (2023) does not guarantee that the normalised embeddings will lie on the hypersphere. The difference in applied distances could result in the miner choosing less optimal pairs as normalised samples may have directions that would violate the CosineSimilarity margin or the Euclidean margin if they were normalised but non-unit lengths, which means they do not violate the Euclidean margin. This opens up an avenue to explore what effect ensuring that normalised embeddings lie on the hypersphere would have on the performance of the XBN approach.

The foundation of XBN acts as a building block from which we explore additional modifications in this work. While XBN suggests it addresses the "slow drift" better than XBM, it is unclear whether further progress can be made in this area. In particular, the per-class and super-class normalised cross-batch memory approaches explored in this work aim to explore this notion.

Chapter 4

Cross-Batch Memory and Variants

This chapter outlines the methodology used in this work, referring to the aim of assessing cross-batch memory and existing variants whilst introducing novel modifications to enhance the efficiency of deep metric learning. We begin by elucidating the standard XBM mechanism, and the XBN modification. We then introduce some proposed modifications, Per-Class Normalised Cross Batch Memory, Super-Class Normalised Cross Batch Memory, and Delayed Normalised Cross Batch Memory. We also explore two additional variants of XBN: Center Normalised Cross Batch Memory and Scaling Normalised Cross Batch Memory.

Experimental parameters and evaluation metrics are detailed in the next chapter, where we present the results of our experiments and discuss the implications of our findings.

4.1 Cross-Batch Memory (XBM)

Having highlighted the difficulty of training deep metric learning models, particularly relating to the computational cost of standard objective functions and the response of Cross-Batch Memory (XBM) to this issue, we seek to provide a detailed overview of the XBM mechanism. In this work, we adopt the XBM mechanism proposed by Wang et al. (2019), which leverages the slow-drift phenomenon observed in the embedding space during training.

4.2 Normalised Cross-Batch Memory (XBN)

While XBM leverages the slow-drift phenomenon, it doesn't explicitly account for the ongoing evolution of the embedding space during training. To address this, XBN introduces a normalisation step to align the stored embeddings with the statistics of the most recent mini-batch.

Specifically, let R denote the reference embeddings in the memory bank and B represent the embeddings generated from the latest mini-batch. XBN normalises the reference embeddings z as follows:

$$\hat{z} = \frac{z - \mathbb{E}[R]}{\sigma[R]}\sigma[B] + \mathbb{E}[B] \tag{4.1}$$

where $\mathbb{E}$ denotes the mean and σ the standard deviation.

This normalszation aims to recalibrate the stored embeddings to better reflect the current state of the embedding space, potentially leading to more informative contrastive learning and improved model performance (Ajanthan et al., 2023).

Algorithm 1: Normalised Cross-Batch Memory Module

Data: xbm_embeddings, xbm_labels,batch_embeddings, batch_labels,
Result: Normalised embedding memory (xbm_embeddings)

```
Function NormaliseEmbeddingMemory():
    batch_mean, batch_std ← mean and standard deviation of
      batch_embeddings
    mem_embeddings ← copy of xbm_embeddings
    mem_embeddings ← normalised mem_embeddings per Eq. 3.6
    xbm_embeddings ← normalised mem_embeddings
```

4.3 Custom Modifications

Building upon the foundation of XBN, we propose three novel modifications designed to further enhance the adaptability and effectiveness of cross-batch memory in DML:

4.3.1 Per-Class Normalised Cross Batch Memory

Instead of normalising the entire memory bank based on the global statistics of the latest mini-batch, we propose a more fine-grained approach: Per-Class normalisation. In this scheme, the normalisation is performed independently for each class, utilizing the class-specific mean and standard deviation computed across both the memory bank and the latest mini-batch.

Mathematically, for class i, let R be the reference embeddings, R^i be the reference embeddings of class i, B be the mini-batch set and B^i be the recent mini-batch embeddings belonging to that class. The normalisation is then expressed in Eq. 4.2

$$\hat{z}_i = \frac{z_i - \mathbb{E}\left[R^i\right]}{\sigma\left[R^i\right]}\sigma\left[B^i\right] + \mathbb{E}\left[B^i\right] \tag{4.2}$$

To provide additional flexibility, we introduce two hyperparameters, λ_μ and λ_σ in Eq. 2, to control the relative influence of the class-specific and mini-batch statistics in the normalisation process.

$$\hat{z}_i = \frac{z_i - \mathbb{E}\left[R^i\right]}{\sigma\left[R^i\right]}\left(\lambda_\sigma \sigma\left[B\right] + (1-\lambda_\sigma)\,\sigma\left[B^i\right]\right) + \left(\lambda_\mu \mathbb{E}\left[B\right] + (1-\lambda_\mu)\,\mathbb{E}\left[B^i\right]\right) \tag{4.3}$$

The introduction of λ_μ and λ_σ are in recognition of the noisy estimates of the first two moments when considering individual classes. The hyperparameters allow for consideration of the class statistics with the backing of the global statistics, providing a more stable and reliable normalisation process as demonstrated by Ajanthan et al. (2023).

For classes absent in the latest mini-batch, two options are available: normalise their embeddings using the global mini-batch statistics (akin to XBN) or leave them unnormalised (as in the original XBM). The choice between these options can be governed by a Boolean flag. We choose to normalise absent classes using global statistics, as it provides a more consistent normalisation process across all classes.

The motivation behind Per-Class normalisation is to capture the nuanced and potentially class-specific dynamics of representational drift, enabling the memory bank to adapt more precisely to the evolving embedding space.

Algorithm 2: Normalised Cross-Batch Memory Module

Data: xbm_embeddings, xbm_labels,batch_embeddings, batch_labels,
Result: Per-Class Normalised embedding memory (xbm_embeddings)

```
Function NormaliseEmbeddingMemory():
    batch_mean, batch_std ← mean and standard deviation of
     batch_embeddings
    unique_labels ← unique labels in latest batch
    for label ∈ unique_labels do
        label_embeds ← xbm_embeddings of class label
        label_queue_embeds ← batch_embeddings of class label
        if length of label_embeds >1 AND length of label_queue_embeds >1 then
            label_embeds ← normalised label_embeds per Eq. 4.3. where R is
             label_embeds, B is label_queue_embeds
    if normalise absent classes then
        other_embeddings ← embeddings of other classes in memory bank
        other_embeddings ← normalised per Eq. 3.6
    xbm_embeddings ← updated label embeddings
```

Note that the embeddings are only normalised for classes with more than one sample in the memory bank and the latest mini-batch. This is to increase stability, but more importantly, ensure variance is non-zero to avoid division by zero.

Early empirical results suggest that a mean lambda of 0.5 and a variance lambda of 1 provide the best performance. These are the values used in the experiments in the next chapter unless stated otherwise.

4.3.2 Super-Class Normalised Cross Batch Memory

Extending from PC-XBN, in scenarios where individual classes have limited sample sizes but belong to broader superclasses with more substantial representation, we introduce Super-Class normalisation. This technique leverages the statistics of the superclasses to normalise the embeddings of their constituent subclasses. This is an attempt to reduce the instability of the normalisation process due to the scarcity of samples per class, which can lead to unreliable summary statistics and degraded performance (Wightman et al., 2021).

This approach is particularly pertinent in few-shot learning settings, where the scarcity of samples per class can hinder effective normalisation at the class level as summary statistics (such as the mean and standard deviation) deteriorate. By operating at the superclass level, we can harness the richer statistical information to guide the normalisation process.

The normalisation transformation is similar to the Per-Class Normalised Cross-Batch Memory but operates at the superclass level. Mathematically, let R^i represent the reference embeddings of superclass i, and B^i denote the recent mini-batch embeddings belonging to that superclass. The normalisation for subclasses within superclass i is then given by:

$$\hat{z}_{subclass} = \frac{z_{subclass} - \mathbb{E}\left[R^i\right]}{\sigma\left[R^i\right]}\sigma\left[B^i\right] + \mathbb{E}\left[B^i\right] \tag{4.4}$$

The implementation is similar to Algorithm 2, adjusting line 3 to get the unique superclasses in the latest batch and lines 4 and 5 to get the embeddings of the superclass and the batch embeddings of the superclass, respectively. The rest of the algorithm remains the same.

While the normalisation occurs at the superclass level, the model's loss and evaluation performance are still evaluated at the subclass level, ensuring that the learned metric space remains discriminative at the finer granularity. In this research, we refer to supervised superclasses as those defined by a dataset label hierarchy. For instance in CIFAR-100, the superclass of "aquatic mammals" would contain the subclasses "beaver", "dolphin", "otter", and "whale" (Krizhevsky, 2009).

Unlike in per-class normalisation, the connection between the semantic superclass and the visual features of its subclasses is not as clear. This may reveal itself in the disconnect between the distribution of the embedding space and the imagined semantic space and therefore the performance of the model; visually dissimilar classes may be detrimental

by normalisation to the same superclass. This may appear to be an inherent issue with the superclass normalisation approach, however, we present that the super class normalisation results in closer semantic clustering at a similar per-class performance to the standard XBM approach. This opens up the possibility of a dual-task application of the superclass normalisation approach, where the model can classify both at the super class and subclass levels.

We propose that similar to the per-class normalisation approach, the super-class normalisation approach uses a mean lambda of 0.5 and a variance lambda of 1 for the best performance. These are the values used in the experiments in the next chapter unless stated otherwise.

4.3.3 Delayed Normalised Cross Batch Memory

Empirical observations and prior research suggest that the most substantial shifts in the embedding space occur during the initial phases of training. To capitalize on this, and similar to the initial XBM delay suggested by Wang et al. (2019) we propose Delayed normalisation, which postpones the normalisation of the memory bank until the model has attained a certain level of convergence. This is similar in premise to the warmup period explored by Wang et al. (2019) and Wang et al. (2021), but with a more explicit focus on delaying the normalisation, not the formation of the memory bank. Delaying the formation of the memory bank will still occur given the epoch warmup period employed by Ajanthan et al. (2023) which will be recreated.

We can very easily delay the normalisation by introducing a hyperparameter, denoted by τ, representing the epoch after which the normalisation commence as normal. Once this epoch threshold is passed, the normalisation proceeds as in the standard XBN approach. The XBM embeddings are thus allowed to evolve freely during the initial training epochs, potentially avoiding premature convergence to suboptimal solutions. Early empirical results suggest that 15 epochs (the first learning rate decay of Ajanthan et al. (2023)) is the optimal delay for the normalisation process. This is the value used in the experiments in the next chapter unless stated otherwise.

The rationale behind delayed normalisation is to allow the model to explore the embedding space more freely in the early stages, potentially avoiding premature convergence to suboptimal solutions. By delaying normalisation, the aim is to strike a balance between leveraging the memory bank's stability and accommodating the dynamic nature of the embedding space during initial training.

4.3.4 Centre Normalised Cross Batch Memory

In addition to the proposed modifications, we explore two additional variants of XBN: Centre Normalisation and Scaling Normalisation. Centre Normalisation involves shifting solely the mean of the embeddings to align with the mean of the latest mini-batch. Whilst similar to the normalisation process in XBN, this approach does not scale the

embeddings by the standard deviation of the mini-batch which is more susceptible to statistical error on reduced sample size.

$$\hat{z} = z - \mathbb{E}[R] + \mathbb{E}[B] \tag{4.5}$$

This is implemented similarly to the standard XBN approach shown in Algorithm 1, except dividing by the standard deviation of the mini-batch. Alg. 1 is therefore updated at line 4 to be the mean of the batch embeddings and remove the standard deviation calculation (Eq. 4.5).

4.3.5 Scale Normalised Cross Batch Memory

Scaling normalisation simply ensures that post the XBN normalisation transformation, the embeddings are normalised to a unit norm. We implement this by simply adjusting the normalisation process in Algorithm 1 to include a division by the norm of the embeddings at line 4.

This would not have an effect on a cosine similarity-based approach, but as outlined in the related work, the `PairMarginMiner` used in the original XBN paper uses a Euclidean distance. This means that the normalisation of the embeddings to a unit norm could affect the performance of the model. An alternative approach would be to switch the distance function used in the `PairMarginMiner` to cosine similarity, however, this would not be a direct comparison to the original XBN paper. Inherently, embeddings in this research are normalised to a unit norm when produced by the model, however, this is not guaranteed to be the case when embeddings are normalised to the statistics of the most recent mini-batch.

Chapter 5

Evaluation

This section outlines the evaluation of the proposed modifications to the Cross Batch Memory algorithm. We evaluate five datasets, including CIFAR-10, CIFAR-100, In-Shop, Stanford Online Products (SOP), and DeepFashion2. We assess in line with Wang et al. (2019), with the Recall@1 and Recall@10 metrics used to evaluate the model's performance where Recall@1 and Recall@10 are the percentage of queries for which the queried class is in the top 1 and top 10 retrieved items, respectively.

In Section 5.1, we outline the five datasets used in the evaluation. In Section 5.2, the software setup used for the evaluation is outlined. The hardware setup is summarised in Section 5.3.

In Section 5.4, we present the overall results of the evaluation.

5.1 Datasets

5.1.1 Stanford Online Products

Stanford Online Products (SOP) (Song et al., 2016) is dataset of 120,053 product images across 22,634 classes, with 2 to 10 photos per class. Following Song et al. (2016) and Ajanthan et al. (2023), the dataset is split into 59,551 images (11,318 classes) for training and 60,502 images (11,316 classes) for testing.

5.1.2 In-Shop

In-shop Clothes Retrieval (IN-SHOP) (Liu et al., 2016) consists of 72,712 clothing images across 7,986 classes. Following Liu et al. (2016) and Ajanthan et al. (2023), we use 25,882 images from 3,997 classes as the training set. We partition the test set into a query set and a gallery set with 14,218 images of 3,985 classes and 3,985 classes with 12,612 photos

respectively. We consider the superclasses to be the category included in the provided `image_name` field such as "Dresses", "Skirts", "Sweatshirts", etc.

5.1.3 DeepFashion2

The DeepFashion 2 (Ge et al., 2019) consumer-to-shop retrieval dataset consists of 217,778 clothing items. Using the provided bounding boxes we crop the appropriate clothing items for training and testing. We use the validation set defined by Ge et al. (2019) for evaluation. The evaluation set consists of 36,961 images in the gallery set and 12,377 in the query set. Following the methodology of Ajanthan et al. (2023), we define a class by the `pair_id`, `category_id`, and `style`, following Ge et al. (2019). There are 31,517, 7,059, and 3,128 classes in the training, gallery and query sets respectively. We consider the superclasses to be the `category_id` in the provided dataset annotations.

5.1.4 CIFAR-10

CIFAR-10 (Krizhevsky, 2009) is a dataset of 60,000 32-by-32 colour images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images.

5.1.5 CIFAR-100

Like CIFAR-10, CIFAR100 (Krizhevsky, 2009) is a dataset of 60,000 32x32 colour images in 100 classes, with 600 images per class. There are 50,000 training images and 10,000 test images. Within the CIFAR-100 dataset, there are 20 provided superclasses.

5.2 Software Setup

All experiments were based on the timm (Wightman, 2019) Pytorch library and associated train.py training script. The ResNet50 model was used as the base model for all experiments with the default pre-trained weights from the timm library (Wightman et al., 2021). We set the embedding dimension to 512 by setting the num-classes parameter to 512.

Default data augmentations for timm were applied, consisting of the centre crop to the appropriate 224-by-224 dimensions for the Resnet50 model, random horizontal flip, and colour jitter. `RandomAugment` was applied with profile 'rand-m9-mstd0.5'. `RandomErasing` was applied with probability 0.2. The models were trained in mixed precision mode, using the `--amp flag`. We clip Gradients to 10.

For metric learning-related functionality, we use the pytorch-metric-learning library (Musgrave et al., 2020b). Similar to Ajanthan et al. (2023) we use a PairMarginMiner with the default parameters $pos\_margin = 0.2$, $neg\_margin = 0.8$. Note that the default distance for the `PairMarginMiner` is Euclidean to replicate the settings produced in the AXBN paper (Ajanthan et al., 2023). `SupConLoss` was used as the loss function for all experiments, with CosineSimilarity being used as the distance metric.

We perform a warmup stage of 2 epochs in line with Ajanthan et al. (2023) on the feature embedding layer. We freeze all other layers during the warmup stage. We set the learning rate to 0.001, and use the `sgd` optimiser. The batch size for all experiments is 64. We use a sampler to ensure that each batch contains at least 4 images from each class.

We perform standard training for 50 epochs with the AdamW optimiser and a learning rate of 0.0001. We use a step learning rate scheduler with a step size of 15 and a decay rate of 0.33. Using Recall@1 and Recall@10 we evaluate the different models and select the the best model based on Recall@1.

The XBM size (i.e. number of stored embeddings) is set to 50% of the training set size with the exception of DeepFashion2 where the size is set to 100% of the training set size to align with Ajanthan et al. (2023). Any parameters specific to particular experiments are detailed in the appendix (Appendix 9).

Complete code for the experiments is available at https://gitlab.cecs.anu.edu.au/u7490752/comp3770-project.

5.3 Hardware Setup

All experiments were run on LambdaLabs GPU cloud service under the profile gpu_1x_a10 with the following specifications:

- 30 vCPU cores
- 1x NVIDIA A10 GPU (24GB VRAM)
- 200 GiB RAM

5.4 Results

This section contains the results and analysis of Cross-Batch memory and various modifications. It is broken down into the following sections:

We start by establishing the baseline performance of the model without any Cross-Batch Memory in Section 5.4.1. We also examine the drift of the embedding space over time. We confirm the slow-drift phenomenon described in Wang et al. (2019) and a general expectation of class drift and clustering within the embedding space.

We evaluate the Cross-Batch Memory algorithm in Section 5.4.2. We find that the XBM algorithm outperforms the No-XBM algorithm across all datasets, replicating the claims of Wang et al. (2019). Just as we did for the baseline, we also analysed the XBM algorithm's drift. We find that XBM slightly increases the drift of the embedding space over time, likely an indicator of a more significant gradient contribution to the loss function and, therefore, more learning due to XBM. We demonstrate this results in a similarly tightly clustered but more separated embedding space.

We then evaluate the Cross-Batch Normalisation algorithm in Section 5.4.3. Surprisingly, the XBN algorithm performs similarly to the XBM algorithm, with a slight decrease in performance on average. An analysis of the embedding space supports XBN's similar performance, which revealed behaviour comparable to that of the XBM approach. Given the similarity to XBN, we also include results for centre-based normalisation and scaling-based normalisation. Unsurprisingly, these methods do not significantly differ performance-wise from XBN.

Introducing PC-XBN as the first of the two novel modifications to XBN in Section 5.4.4, we find that the PC-XBN algorithm performs significantly worse than the XBN algorithm. This is despite better clustering of the embeddings of a given class, which leads to insights into the importance of miners in identifying hard pairs.

Finally, we explore SC-XBN and find significantly more positive results than PC-XBN in Section 5.4.5.

In all sections, we provide task-based results, namely Recall@1 and Recall@10, as well as analysis of the average drift using Ajanthan et al. (2023)'s drift metric in Eq. 3.5, the average distance between embeddings, and the average distance from an embedding's class mean. The average distance between embeddings should be used to gauge how separate they are in the embedding space. The average distance from an embedding's class means should be used to gauge how well the model clusters embeddings of the same class.

Full results from all experiments are available in the appendix.

5.4.1 No-XBM Baseline

The study is commenced at the most basic level by evaluating the performance of the No-XBM model. The results are shown in Table 5.1.

Table 5.1: Results of the XBM algorithm on the In-Shop, SOP and DeepFashion2 datasets.

	In-Shop	**SOP**	**DeepFashion2**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10	Recall@1 | Recall@10
No-XBM	47.67 | 76.62	44.68 | 60.83	24.50 | 43.06

Something immediately of note is the significantly reduced performance of the baseline No-XBM model when compared to the results reported by Ajanthan et al. (2023), our comparison study. The most notable difference is the performance across the No-XBM baselines. In the three datasets, the previous results were 75.94, 88.76 and 33.45 respectively, compared to the 45.0, 47.2 and 24.5 reported here. This is admittedly a significant reduction in performance. Despite several conversations with two of the authors of the XBN paper, it was not possible to determine the cause of this discrepancy. Experimental parameters, as significant as the model structure to the batch size and learning rate, were kept consistent with the original paper. Dataset splits were verified to be performed using the same method, and the same datasets were used. There have been changes to both the timm library and the pytorch-metric-learning library since the original paper was published, which may have contributed to the difference in results. This does not invalidate these baseline results or perhaps more importantly the results of the following experiments. The primary goal of this work is to assess the impact of the proposed normalisation approaches on the embedding space. The reduced performance of the No-XBM model is unlikely to change the relative performance of the XBM, XBN, PC-XBN and SC-XBN models given they use the same foundations. As such, further experiments should be analysed relative to the No-XBM model, not objectively. The results of these models are still valid and provide insight into the impact of the proposed normalisation approaches on model performance and the embedding space.

From the training curves of the models in Appendix 8 it appears that some models were unable to train in the No-XBM context with performance peaking in the first few epochs and reducing from there. This is particularly apparent in the SOP (Fig. 8.4) and DeepFashion (Fig. 8.5) datasets which demonstrate decreasing Recall@1 performance over time. Poor training of these datasets is highlighted in the average distance from the class mean in Fig. 5.3 where SOP and DeepFashion demonstrate an increase in distance from the class means over time. It is worth noting these datasets are significantly larger than the other evaluation datasets and both have small per-class sample sizes. All of these factors indicate that the supervised contrastive loss cannot effectively create a discriminative embedding space for these datasets. As previously outlined, the performance of the loss function is dependent on the quality of the pairs, or more specifically the

hard pairs, that are provided by the miner (Harwood et al., 2017) (Wang et al., 2019). The poor performance, both in Recall and in the embedding space of these datasets can therefore likely be attributed to a lack of hard pairs. Consequently, increasing access to hard pairs and thereby the gradient contribution to the loss function would likely assist in increasing the separability of the embedding space, presenting a major justification for the introduction of the XBM algorithm (Wang et al., 2019).

Moving away from the evaluation metrics, we confirm the first justification for cross-batch memory is the notion of slow drift; the embedding space only drifts significantly in the first few epochs of training. We confirm this in 5.1, where we see a decrease in drift over time. We also find justification for the warmup period or delay in the use of the memory bank in the first few epochs of training in the brief increase in drift at the start of training. This period of increasing drift is indicative of the model moving towards a more stable embedding space. We highlight there are two noticeable reductions in the drift at epoch 15 and epoch 30, with a more minor reduction at epoch 45. These drops match the scheduled decay of the learning rate.

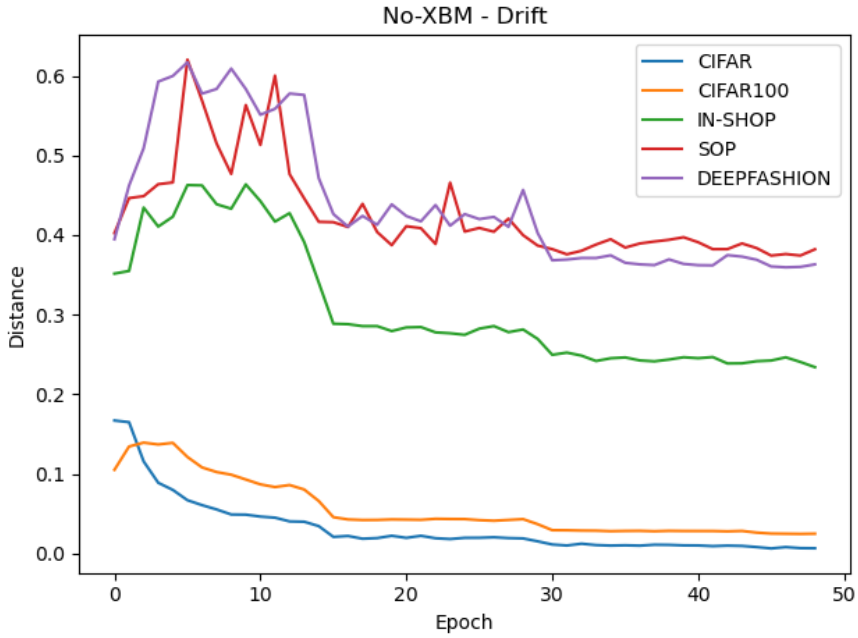

Figure 5.1: Average drift of class embeddings over training epochs

Excluding the DeepFashion and SOP trials, we find the embedding space becomes more separable as training progresses in Fig. 5.2 and Fig. 5.3. The average distance between embeddings increases over time, showcasing the learning of the model to separate embeddings of different classes.

Further, we see classes are drawn together in the embedding space by the Supervised Contrastive Loss, as shown in Fig. 5.3. The average distance from an embedding's class

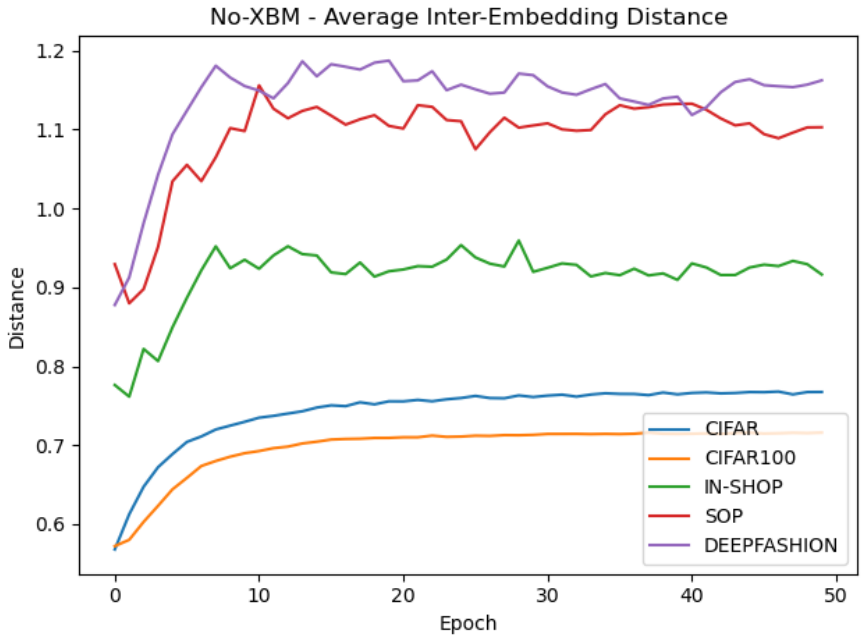

Figure 5.2: Average distance between embeddings

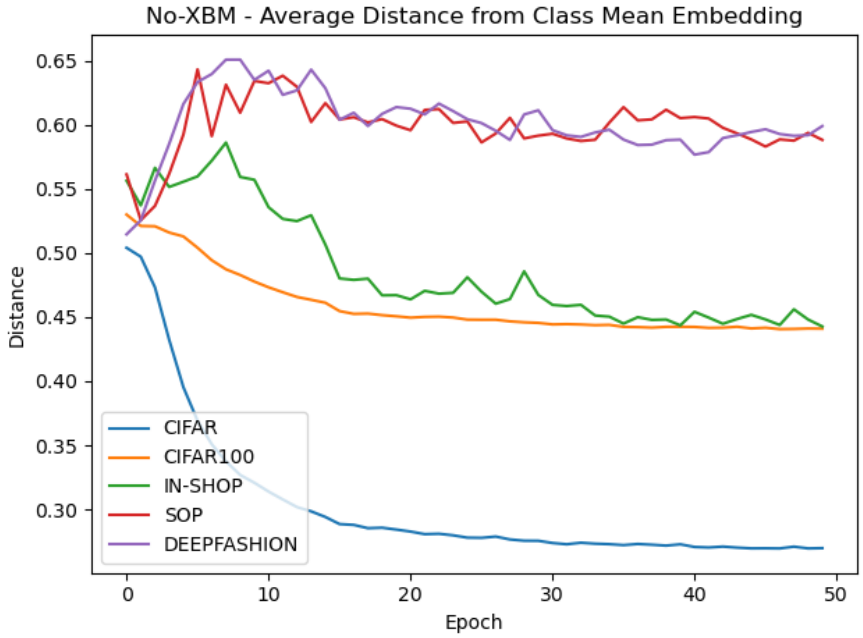

Figure 5.3: Average distance from class mean - CIFAR-100

mean decreases over time, showcasing the learning of the model to cluster embeddings of the same class. The effect of slow drift is apparent in both measures with the rate of change decreasing over time, particularly post the first learning rate decay at epoch 15.

Whilst not the primary focus of this work, the explored baseline results provide a strong foundation for the evaluation of XBM and further normalisation approaches. Importantly, they highlight that even without the presence of XBM, the movement of the embedding space towards its highly separable state occurs through the separation of embeddings of different classes and the clustering of embeddings of the same class. This is a key insight into the behaviour of the embedding space and the impact of the proposed normalisation approaches.

5.4.2 Cross Batch Memory

This section evaluates the Cross Batch Memory (XBM) algorithm. With the background of the poor training behaviour and performance of No-XBM models alongside promising results of cross-batch memory in literature, the expectation that the XBM algorithm significantly outperforms the No-XBM algorithm is confirmed. The results are shown in Table 5.2.

Table 5.2: Results of the XBM algorithm on the In-Shop, SOP and DeepFashion2 datasets.

	In-Shop	**SOP**	**DeepFashion2**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10	Recall@1 | Recall@10
No-XBM	47.67 | 76.62	44.68 | 60.83	26.50 | 43.06
XBM	**71.68** | 91.12	**58.73** | 74.11	**43.63** | 62.57

The XBM algorithm outperforms the No-XBM algorithm across the board. Similarly to the outlined reduced baseline performance, the performance of the XBM approach appears to be significantly reduced when compared to the results reported by Wang et al. (2019) of 89.1 for In-shop and 77.8 for SOP. We find evidence of the disproportionately weak baseline affecting this result in the more significant relative performance improvement compared to Wang et al. (2019) with the XBM algorithm, 28.9% and 51.9%, respectively. A hypothesis for this comparatively better result is the increased memory bank size and, therefore, increased opportunity to find optimal pairs employed in this work.

Additionally, XBM is evaluated on CIFAR-10 and CIFAR-100 datasets with the expectation of further positive results met. This is despite the significantly reduced number of classes in the CIFAR datasets, which leads to more stale embeddings per class stored in the cross-batch memory and a possible negative impact on the relative performance. The results are shown in Table 5.3. We see a slight reduction in the relative performance increase between baseline and XBM approaches, with a 25.9% and 6.6% improvement for CIFAR-10 and CIFAR-100, respectively. Whilst this may indicate that the XBM algorithm is less effective on smaller datasets, a more likely argument is that the reduced number of classes in the CIFAR datasets requires a less granular embedding space to achieve optimal performance. XBM's ability to assist in increasing the granularity of the embedding space is, therefore, less impactful. The analysis of the embedding space supports this argument with tighter inter-class distances demonstrated by XBM compared to No-XBM in Fig. 5.6

Whilst empirically demonstrated in both this thesis and previous results, the XBM algorithm's success is attributed to the increased hard pairs accessible to the model. This is supported by the drift analysis of the XBM algorithm, shown in Figure 5.4. We observe quite a significant increase (2x) in drift over the training period, particularly in the initial stages, when compared to the No-XBM results. This demonstrates the increased learning

Table 5.3: Results of the XBM algorithm on the CIFAR-10 and CIFAR-100 datasets.

	CIFAR-10	**CIFAR-100**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10
No-XBM	**96.95** | 98.98	77.24 | 91.11
XBM	96.85 | 98.86	**82.01** | 91.68

rate of the model, which is a direct result of greater gradient contributions to the loss function through the increased hard pairs accessible to the model. Other features of the drift are reasonably similar to the No-XBM algorithm.

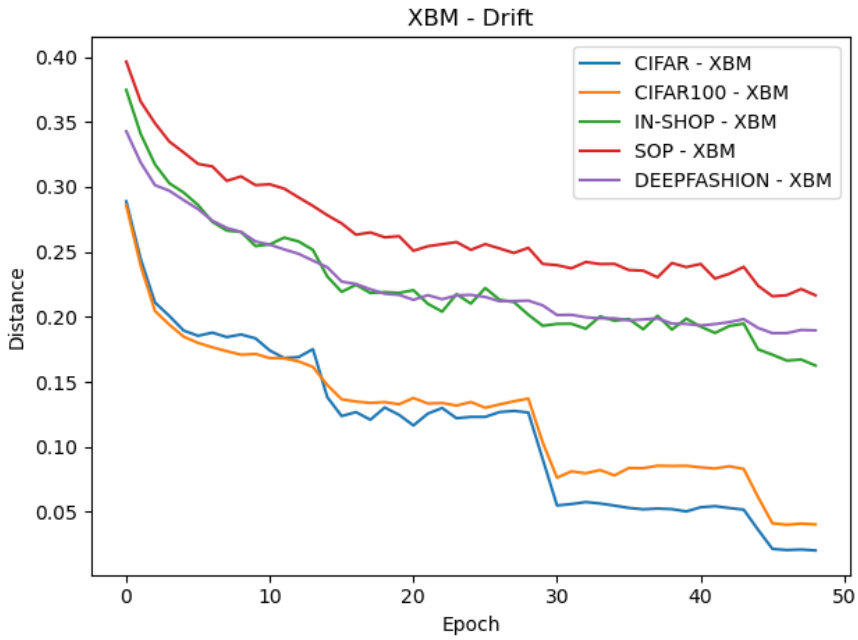

Figure 5.4: XBM Drift over time

As mentioned above, we empirically demonstrate the ability of the XBM algorithm to increase the granularity of the embedding space in the analysis of the embedding space movement shown in Figures 5.5 and 5.6.

We find a very slightly increased average distance between embeddings and similarly tight clustering of classes by XBM compared to the No-XBM algorithm. This reveals that the increased Recall@1 performance likely comes from the increased separation of differing classes, not tighter clustering of the same class. Considering that the No-XBM approach has already demonstrated a relatively tight clustering of the same class, we can infer that XBM increases access to hard negative pairs, not hard pairs in general.

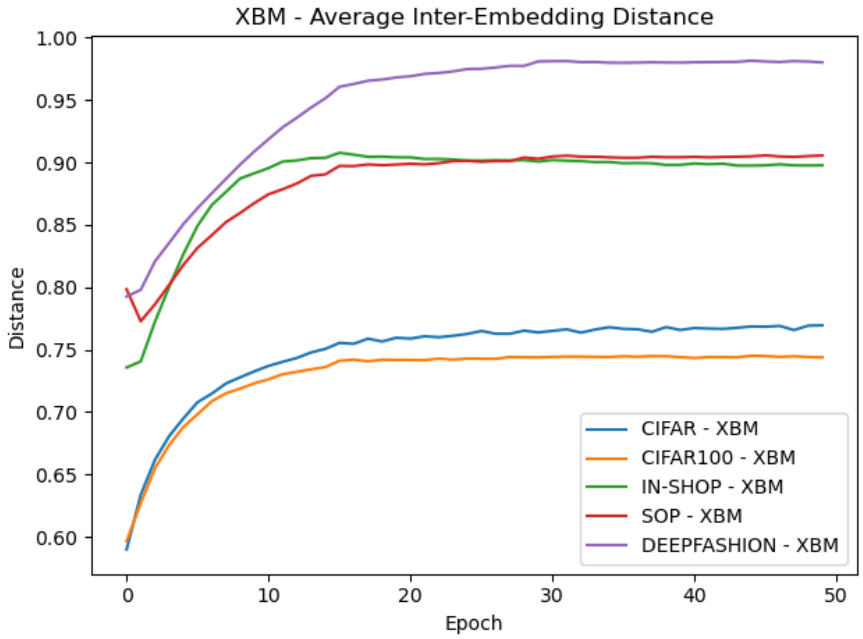

Figure 5.5: Average XBM Inter-embedding distances over time

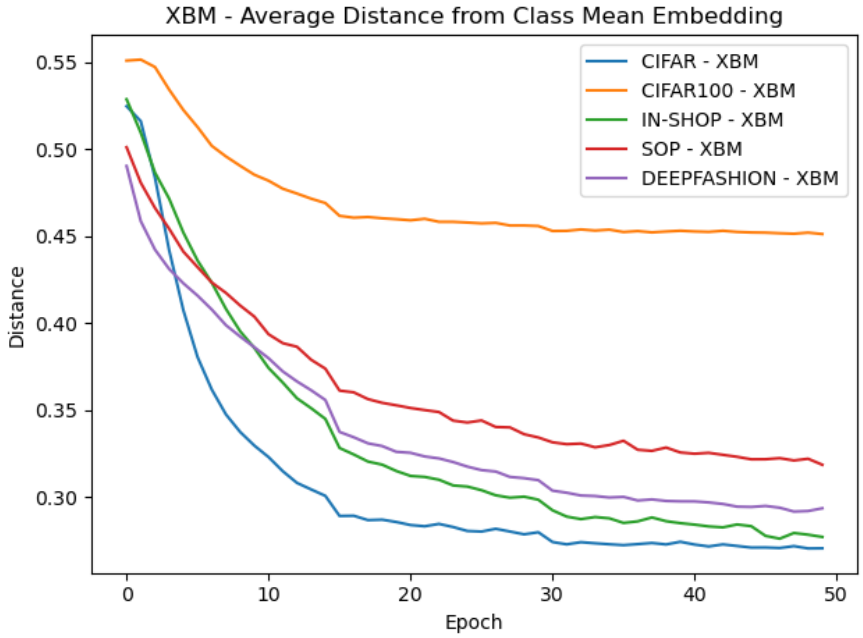

Figure 5.6: Average XBM distance from class mean over time

Ultimately, we confirm the success of the XBM algorithm in increasing the granularity of the embedding space and, therefore, the model's performance. It has been demonstrated empirically that this is achieved through the increased hard pairs accessible to the model in the drift analysis and the increased average distance between embeddings, further supported by the increased Recall@1 performance across all datasets.

5.4.3 Normalised Cross Batch Memory

Having established the impact of the XBM algorithm on the embedding space it is desirable to understand how significantly normalisation, with the aim of addressing the still present drift of slow-drift, can improve the performance of the model. We find that despite previous work suggesting that normalisation can significantly improve the performance of the model, our results are less convincing. The results are shown in Table 5.4.

Table 5.4: Results of the XBN algorithm on the In-Shop, SOP and DeepFashion2 datasets.

	In-Shop	**SOP**	**DeepFashion2**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10	Recall@1 | Recall@10
No-XBM	47.67 | 76.62	44.68 | 60.83	24.50 | 43.06
XBM	**71.68** | 91.12	**58.73** | 74.11	**43.63** | 62.57
XBN	71.02 | 90.95	58.63 | 73.78	43.53 | 61.33

Surprisingly, on the three larger datasets, the XBN algorithm underperforms expectations. Across the board XBM betters XBN, which is in contrast to the results reported by Ajanthan et al. (2023). The performance difference is not particularly significant, with an average decrease of 0.2% across the three trials. A similar trend is seen in the results of the smaller datasets in Table 5.5.

Table 5.5: Results of the XBN algorithm on the CIFAR-10 and CIFAR-100 datasets.

	CIFAR-10	**CIFAR-100**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10
No-XBM	**96.95** | 98.98	77.24 | 91.11
XBM	96.85 | 98.86	**82.01** | 91.68
XBN	96.90 | 98.83	81.91 | 91.69

It does appear that the smaller datasets respond more positively to XBN than the larger datasets, with CIFAR-10 being the only dataset to report an increase in performance. An immediate conclusion is that the greater samples per class of these datasets may be assisting in creating more robust class statistics for the normalisation process.

The XBN approach does not signficantly change the trend of drift of the embedding space in Figure 5.7 when compared to XBM. Although it does significantly increase the amount of drift that occurs. In some instances, XBN results in a 2x increase in drift at some points over the training period. The probable explanation for this is the active nature of the normalisation process which consistently changes the distribution of the reference set, whether accurately or inaccurately, which leads to more drift in the embedding space. If it is assumed that the drift is a result of accurate normalisation,

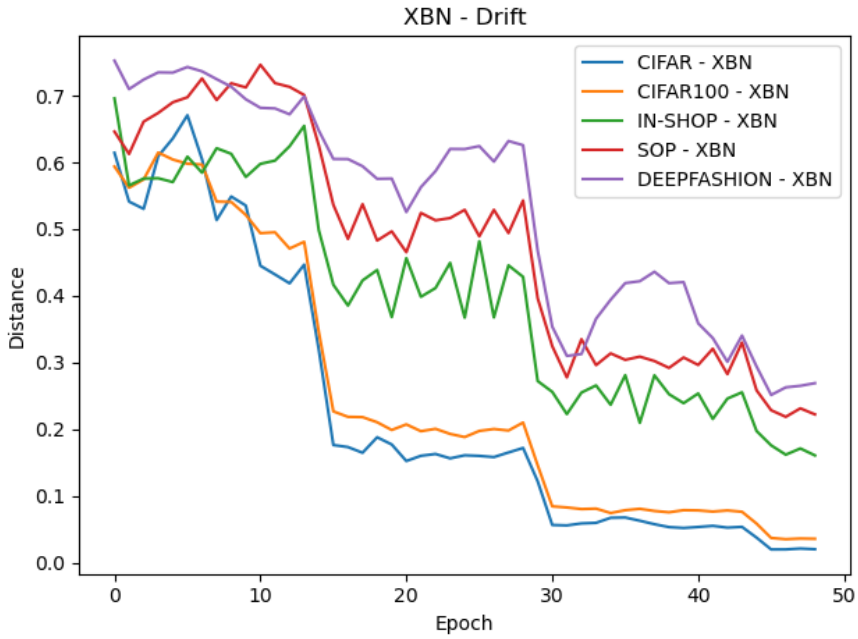

Figure 5.7: XBN drift over time

then the increased drift indicates that the space is moving more consistently and more quickly towards a more stable embedding space. This hypothesis is not supported by the results of the XBN algorithm, which underperforms the XBM algorithm. Instead, if it is assumed the normalisation is not completely accurate we would likely get more violating pairs, which would lead to more drift through the increased gradient contributions to the loss function. The latter hypothesis is more likely, and is supported particularly by the increased drift at the start of training, where the model is attempting to move towards a more stable embedding space and therefore likely to have a non-optimal embedding distribution.

Unlike the XBM algorithm, the XBN algorithm only demonstrates a very slight increase in the average distance between embeddings over time. Most notably, the decrease until the end of training post the sharp increase in distance in the early epochs is not present in the XBN and No-XBN experiments. Despite the general decrease in the distance between embeddings over the training, the end result is still an increase in distance, and thereby the separability of the embedding space. This might explain why the XBN algorithm performs very similarly to the XBM algorithm.

Overall, we find that normalisation does not significantly improve the performance of cross-batch memory. This is at odds with the claimed performance improvements of Ajanthan et al. (2023). Performance is more promising in the smaller datasets, which confirms the normalisation process is likely to be dependent on the quality of the class statistics. This provides justification for both the adaptive normalisation approach via

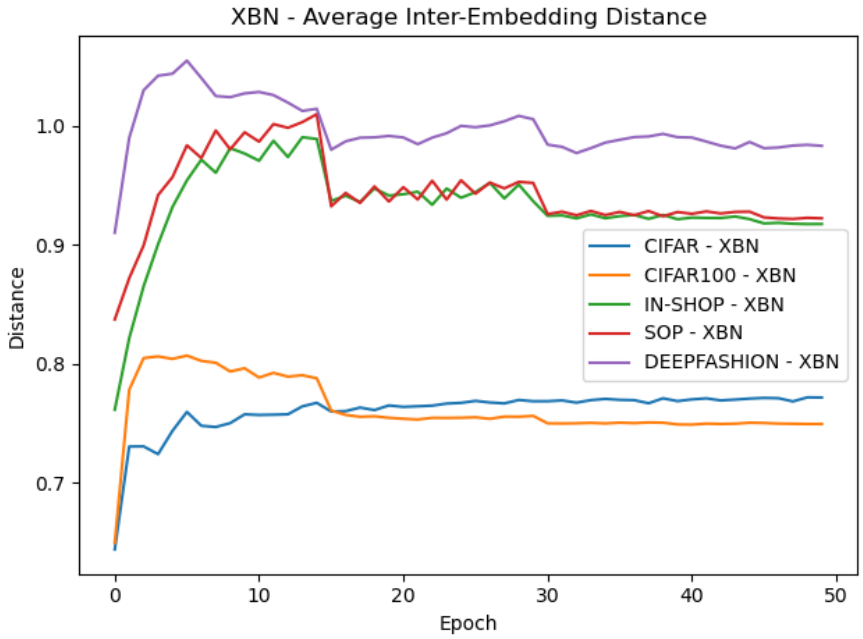

Figure 5.8: Average XBN Inter-embedding distances

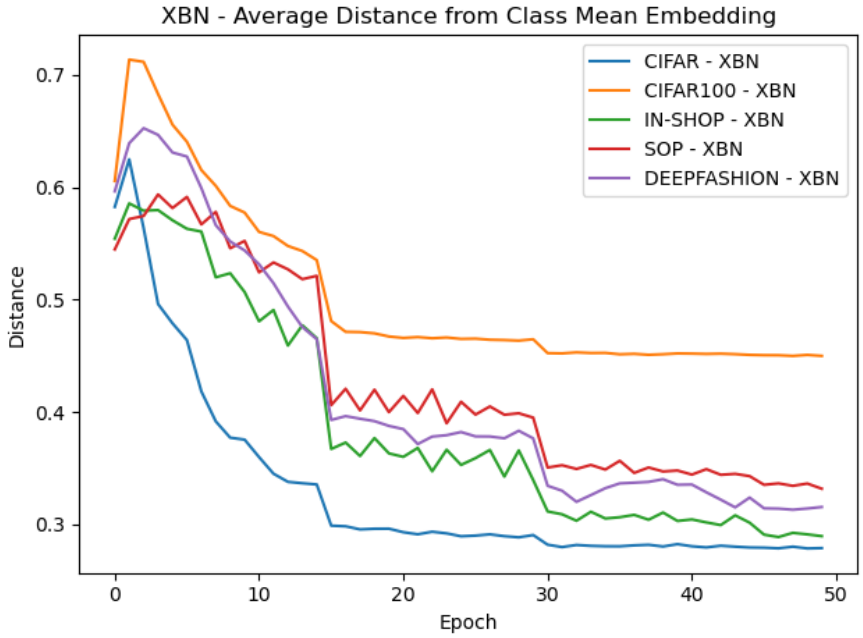

Figure 5.9: Average Intra-class distances over time

Kalman filter suggested by Wang et al. (2019) and an increased granularity normalisation approach, such as the per-class normalisation approach explored later in this work.

Delayed Normalised Cross Batch Memory

Given the increased volatility of the embedding space in the first epochs of XBN training, we investigate the impact of delaying the normalisation process. Overall, delaying the normalisation of the memory bank by 15 epochs has a net positive impact on the performance of the XBN algorithm. The results are shown in Table 5.6. Unfortunately, Recall@10 results were not recorded for the delayed normalisation approach, so only Recall@1 results are shown.

Table 5.6: Results of the XBN algorithm with a 15 epoch delay

	In-Shop	**SOP**	**DeepFashion2**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10	Recall@1 | Recall@10
XBM	**71.68** | 91.12	58.73 | 74.11	43.63 | 62.57
XBN	71.02 | 90.95	58.63 | 73.78	43.53 | 61.33
XBN (15 epoch delay)	71.50 | -	**58.90** | -	**50.60** | -

Table 5.7: Results of the XBN algorithm with a 15 epoch delay

	CIFAR-10	**CIFAR-100**
Algorithm	Recall@1 | Recall@10	| Recall@10
XBM	96.85 | 98.86	**82.01** | 91.68
XBN	96.90 | 98.83	81.91 | 91.69
XBN (15 epoch delay)	**96.90** | -	82.00 |

In all but one experiment, the delayed normalisation approach outperforms XBN. The most notable improvement is on the CIFAR-10 dataset, where the delayed normalisation approach outperforms XBN by 0.5%. This is a significant improvement, given the relatively high performance of the XBN algorithm on the CIFAR-10 dataset. Interestingly, this discovery too is somewhat contradictory to the justification behind XBN, which is that normalisation better helps the possible stale memory bank reflect the current embedding space. When considering the reasoning for the improved performance of the delayed normalisation approach there are two possible related hypotheses. The first is that the normalisation process can cause the training process to get stuck in a local minimum. Both Ajanthan et al. (2023) and Wang et al. (2019) recognise the potential for XBM to cause the model to become stuck in a local minimum, and suggest the use of a warmup period to address this issue. The delayed normalisation approach can be seen as an extension of this warmup period. The requirement for a longer warmup period for normalisation may be due to the active nature of the normalisation process, directly working on the embedding space compared to the passive nature of the memory

bank in XBM. The second possible explanation is that the summary statistics used in the normalisation process are unstable in the early stages of training due to the small number of samples in the memory bank. Whilst the mini-batch size, and therefore the quality of the batch mean, remains constant throughout training, the memory bank size increases over time until it is full. This notion is touched on by Ajanthan et al. (2023), who recognise the effect that instability in the summary statistics can have on the normalisation process and suggest the implementation of a Kalmaan filter to address this issue. Interestingly, we find that the datasets with the highest performance in the delayed normalisation approach are those with the lowest number of classes.

Centre Normalised Cross Batch Memory

Overall, the centre normalised cross batch memory performed very similarly to the standard XBN algorithm. The results are shown in Table 5.8.

Table 5.8: Results of the Centre Normalised XBN algorithm on the In-Shop, SOP and DeepFashion2 datasets.

	In-Shop	**SOP**	**DeepFashion2**
Algorithm	Recall@1	Recall@1	Recall@1
XBN	71.02	58.63	**43.53**
Centre Normalised XBN	**71.57**	**58.96**	43.5

Table 5.9: Results of the Centre Normalised XBN algorithm on the CIFAR-10 and CIFAR-100 datasets.

	CIFAR-10	**CIFAR-100**
Algorithm	Recall@1	Recall@1
XBN	**96.90**	81.91
Centre Normalised XBN	96.81	**81.97**

This is not particularly surprising given the similarities in the transformation of centreing and XBN. These results suggest that the variance is not a significant factor in the normalisation process in terms of improving the performance of the model. Instead, it likely adds a level of complexity to the normalisation process that is not necessary.

Scaling Normalised Cross Batch Memory

The scaling normalised cross batch memory approach performed similarly to the standard XBN algorithm, leaning towards a slight increase in performance.

The resuls in Tables 5.10 and 5.11 show that there is a slight improvement in performance when normalising the embeddings to a unit norm on all datasets except CIFAR-100.

Table 5.10: Results of the Scaling Normalised XBN algorithm on the In-Shop, SOP and DeepFashion2 datasets.

	In-Shop	**SOP**	**DeepFashion2**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10	Recall@1 | Recall@10
XBN	71.02 | 90.95	58.63 | 73.78	43.53 | 61.33
Scaling Normalised XBN	**71.35** | -	**58.64** | -	**49.66** | -

Table 5.11: Results of the Scaling Normalised XBN algorithm on the CIFAR-10 and CIFAR-100 datasets.

	CIFAR-10	**CIFAR-100**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10
XBN	96.90 | 98.83	**81.91** | 91.69
Scaling Normalised XBN	**96.9**| -	81.7 | -

As the `SupConLoss` function makes use of a cosine similarity distance function, the only differentiating factor is the `PairMarginMiner`'s use of the Euclidean distance as explained in the related work. The slightly increased performance suggests that the distance function does impact the optimality of the pairs are selected as hard pairs and thereby the learning of the model. A fair conclusion to make is that cosine similarity should be employed in the miner or the memory bank should be scaled when performing the normalisation process.

5.4.4 Per-Class Normalised Cross Batch Memory

Having established the impact of the XBM algorithm and the potential issues of normalisation to improve the model's performance, we now investigate the impact of normalising embeddings on a per-class basis. Whilst theoretically promising, the Per-Class Normalised Cross Batch Memory approach did not produce positive results when evaluated under the standardised training process. The results are shown in Table 5.12.

Table 5.12: Results of the PC-XBN algorithm on the In-Shop, SOP, DeepFashion2, CIFAR, and CIFAR-100 datasets.

	In-Shop	**SOP**	**DeepFashion2**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10	Recall@1 | Recall@10
Per-Class XBN	10.04 | 25.71	41.76 | 57.71	11.18 | 22.55

Table 5.13: Results of the PC-XBN algorithm on the CIFAR-10 and CIFAR-100 datasets.

	CIFAR-10	**CIFAR-100**
Algorithm	Recall@1	Recall@1
Per Class XBN	85.69 | 96.67	39.69 | 77.36

It is apparent that most models, apart from those trained on the CIFAR-10 and CIFAR-100 datasets, could not train as shown in Chapter 8. Even the CIFAR-10 and CIFAR-100 models performed poorly, ranking below No-XBM baselines. Given that the CIFAR dataset has a much-reduced number of classes compared to the other datasets, we can conclude that per-class normalisation is incredibly sensitive to the number of samples for a given class available in the minibatch and memory bank. This conclusion is not particularly surprising, as the per-class normalisation approach is designed to normalise embeddings based on the class mean. A small number of samples for a class poorly estimates the mean, leading to poor normalisation. Wang et al. (2019) recognise this issue by implementing a Kalman filter as an extension to their normalisation in recognising the noisy estimates of the first two moments, even when considering the entire batch.

One empirical indicator of poor performance is that the drift of the PC-XBN is much less stable than the previous XBN and XBM approaches. Across the training period, the drift is erratic, as shown in Figure 5.10.

An obvious observation from Fig. 5.11 is that the ultimate average embedding distance is significantly lower than the standard XBN approach and lower than that of the XBM approach. A reduced inter-embedding means a diminished ability to discriminate between embeddings and explains the significantly reduced performance of the Per-Class XBM approach.

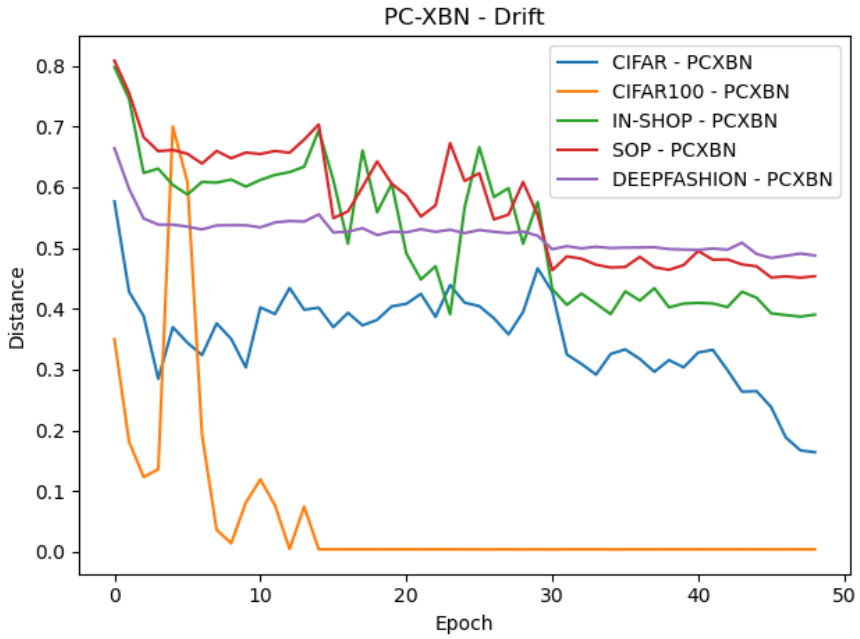

Figure 5.10: PC-XBN drift over time

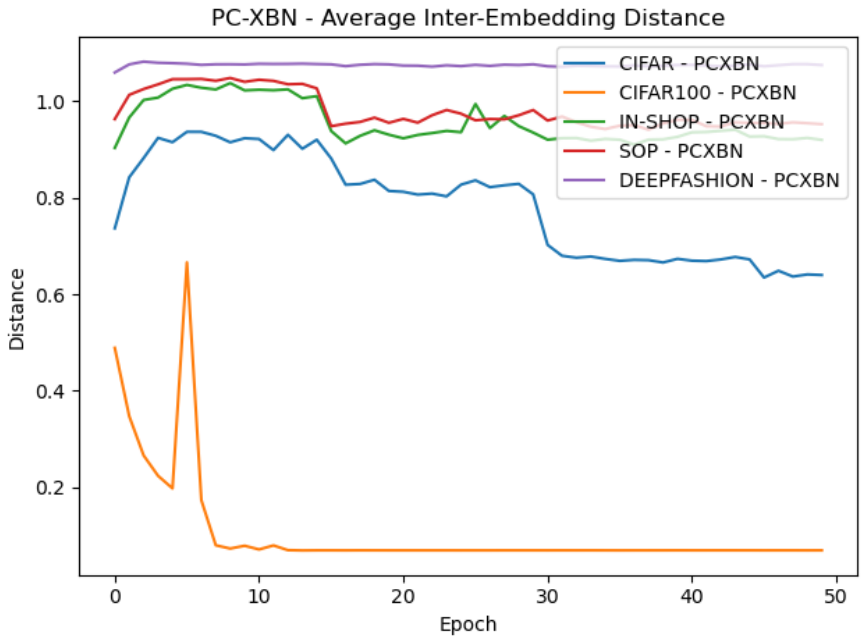

Figure 5.11: Average Inter-embedding distances over time

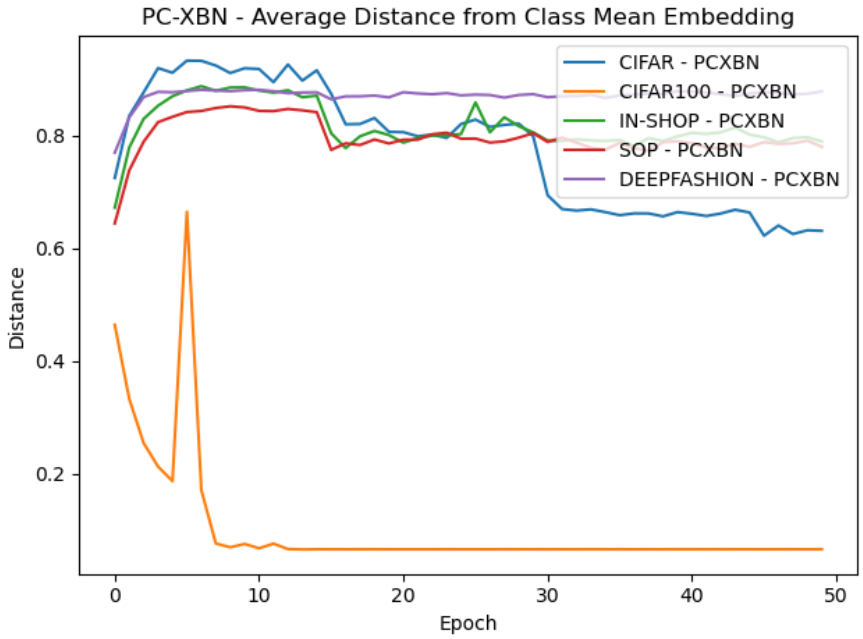

Figure 5.12: Average Intra-class distances over time

Furthermore, the average distance from an embedding's class mean also trends higher than that of the XBN and XBM approaches, as shown in Figure 5.12. The higher inter-embedding distance is likely a result of the poor normalisation of the embeddings, leading to a reduced ability to draw embeddings of the same class together. A higher inter-embedding distance is a significant issue as the Supervised Contrastive Loss draws embeddings of the same class together and pushes embeddings of different classes apart. The poor normalisation of the embeddings leads to a reduced ability to learn from the Supervised Contrastive Loss and, therefore, a reduced performance. The poor separation of the embedding space is likely the primary reason for the poor performance of the Per-Class XBN approach.

The poor quality of normalisation statistics was not surprising, as having identified this issue from previous work we proposed the introduction of the hyperparameters λ_μ and λ_δ in Eq. 3.6 to improve the quality of the first two moments. We conducted experiments on the CIFAR-100 dataset to assess the impact of these lambdas on the performance of the Per-Class XBN approach. Ideally, moving towards the batch mean and variance would lead to a more stable normalisation and, therefore, improved performance. This is true for the variance lambda, but less for the mean lambda as shown in Table 5.14.

The results show that λ_μ significantly impacts the performance of the Per-Class XBN approach. The best performance is achieved with a mean lambda of 0.6 and a Recall@1 of 91.7%. This is a significant improvement over the standard Per-Class XBN approach. Unsurprisingly, given its likely instability, the best performance is achieved with a mean

Table 5.14: Results of the PC-XBN algorithm with mean and variance lambdas on the CIFAR-100 dataset.

	0.0	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
Lambda	Recall@1	Recall@1	Recall@1	Recall@1	Recall@1	
μ	72.5%	74.9%	84.8%	91.7%	60.4%	63.9%
δ	73.5%	73.4%	73.2%	73.5%	73.8%	85.0%

lambda that only partially relies on the class mean. The best performance is not achieved with a λ_μ of 1.0, which would rely solely on the batch mean. In search of an explanation, returning to the normalisation transformation in Eq. 3.6, it can be observed that we conduct initial z-score normalisation on the statistics of the memory bank, whereas we conduct PC-XBN on the statistics of the class (Eq. 4.3).

In the case that $\lambda_u = 1.0$, the PC-XBN transformation is, therefore, across two different distributions, which might explain the reduced performance. An initial normalisation of the statistics of the memory bank followed by the projection to the class statistics would be a more stable approach. This is still a transformation between different distributions, but it may be an avenue for further exploration.

Compared to the λ_μ, λ_δ, however, has a much smaller impact on the performance of the Per-Class XBN approach. The best performance is achieved with a variance lambda of 1.0, meaning that the batch variance is used rather than the class variance, with a Recall@1 of 85.0%. Relying wholly on the batch statistic is more expected than the behaviour of λ_μ and highlights how the variance is much more susceptible to reduced sample size and degraded performance.

When conducting the PC-XBN experiments, it became clear that it severely impairs the training process. It was discovered that, in some cases, the training loss was reduced to zero, which would pose a significant issue for the training process. It soon became evident that the PairMarginMiner was likely decreasing the performance of the per-class normalisation approach. As the experimental setup outlines, the PairMarginMiner has a positive margin of 0.2 and a negative margin of 0.8. This margin is the same as the standard XBN approach (Ajanthan et al., 2023). However, the Per-Class XBN approach demonstrates a reduced distance between embeddings of the same class. As a result, the PairMarginMiner with default margin parameters would often be unable to find any (or a significant number) positive pairs. The SupConLoss construction shown in Eq. 3.3 places the positive samples in the numerator, resulting in a loss of 0 when there were no, or very few, positives and, therefore, no learning. This suggests that the per-class normalisation leads to reduced distance between embeddings of the same class, which is good. However, the controlled setup meant the PC-XBN approach was significantly impaired.

There are a few approaches to address this issue. The most straightforward solution is

to reduce the margin of the PairMarginMiner, thereby allowing for more positive pairs to be found. A significant downside to this approach is that it would likely decrease the optimality of the selected pairs, where the desired pairs violate the margin the most Schroff et al. (2015). Despite this, we assessed whether reducing the positive margin would improve the performance of the Per-Class XBN approach. We found that PC-XBN significantly enhanced the performance of the Per-Class XBN approach and did not degrade as training progressed. Another approach would be to remove the miner entirely. Removing the miner is similar to reducing the margin as maximally possible, as we consider all pairs. Whilst this approach would undoubtedly increase the number of pairs considered, there are significant practical concerns. One such concern is that the number of possible pairs is vast for datasets of considerable size. However, depending on the hardware available, this would substantially increase the time taken to train the model or possibly even make training infeasible.

Table 5.15: Results of the PC-XBN algorithm with adjusted margins on the CIFAR-10 and CIFAR-100 datasets.

	CIFAR-10	**CIFAR-100**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10
PC-XBN	85.69 | 96.67	39.69 | 77.36
PC-XBN (Adjusted Margin)	**96.22** | 98.49	**80.07** | 92.31

To achieve the results of Table 5.15, we also increased the negative margin to 2.0, such that the miner considers all negative pairs; the full margin adjust parameters can be found in Appendix 9. We find a significant performance improvement, bringing the performance of the PC-XBN method to just below the standard for XBM approaches on the CIFAR and CIFAR-100 datasets. These results show considerable improvement but still do not match the performance of other normalisation approaches. The embedding space also does not show the same level of separation as the XBM approach, as shown in Fig. 5.14 and Fig. 5.15. Whilst the inter-embedding distance is signficantly increased on the CIFAR-100 dataset this may be due to the adjusted margins. The drift of the PC-XBN approach is also significantly higher than the XBM approach, as shown in Fig. 5.13 with a signficantly noisier drift pattern. Furthermore, the benefits of the margin adjustments did not extend to the In-Shop, SOP and DeepFashion2 datasets. As outlined above, the larger datasets have significantly reduced number of samples per class in the minibatch and the memory. This poor estimation of the class mean and variance leads to poor normalisation of the embeddings, and therefore, adjusting the margin does not address the poor performance.

Ultimately, the Per-Class Normalised Cross Batch Memory approach did not improve the model's performance. Again, the quality of the normalisation statistics remains a crucial concern when extending cross-batch normalisation. The poor performance of PC-XBN is particularly apparent in the larger datasets with low samples per class meaning PC-XBN is not a viable approach for similar datasets. In this result, we find justification

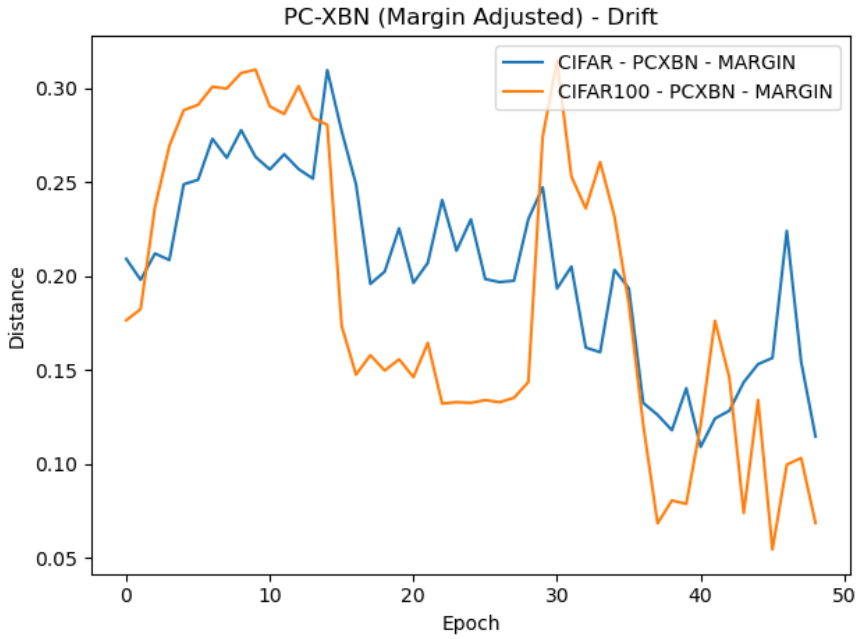

Figure 5.13: PC-XBN (Adjusted Margin) drift over time

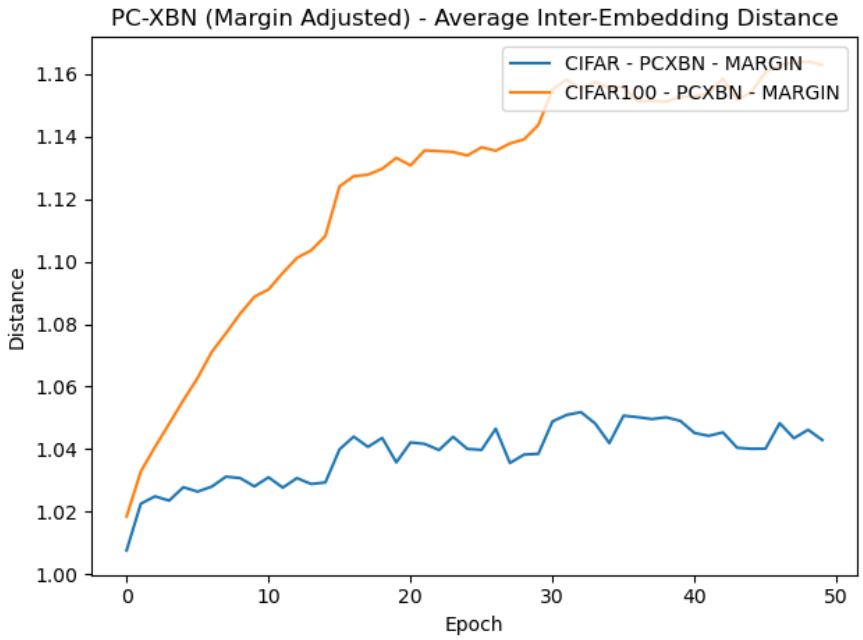

Figure 5.14: Average PC-XBN (Adjusted Margin) Inter-embedding distances over time

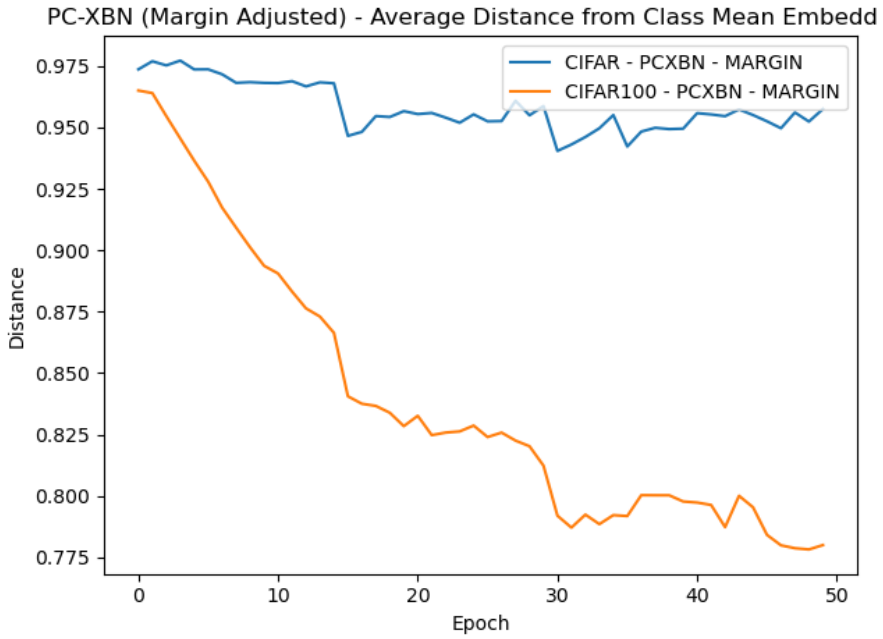

Figure 5.15: Average PC-XBN (Adjusted Margin) Distance from class mean over time

for a less granular normalisation of the embedding space, such as the SC-XBN approach explored in the next section. Despite the somewhat negative result of PC-XBN, there are some positive takeaways; with the right parameters we can achieve performance on the level of XBM. We suggest that there may be an optimal setup for the PC-XBN approach that can improve the model's performance, whether by dataset or by hyperparameter tuning. Finding the ideal setting is a potential avenue for future work.

5.4.5 Super-Class Normalised Cross Batch Memory

We start by investigating how the embedding space relates to superclasses. Given the incompatibility of the larger datasets with PC-XBN established in the previous section, we focus on the CIFAR-100 dataset and its 20 superclasses. As the superclass means are the targets for normalisation, we would like the embeddings of the subclasses to be relatively close to their superclass means so that the normalisation process can be effective. To this end, we examine the average distance of embeddings from their superclass means across the previous experiments. We use the average distance to measure the proportion of embeddings to which their ground truth superclass is in their top k nearest superclass means. The results are shown in Table 5.16.

Table 5.16: Proportion of embeddings whose superclass is in their top k nearest superclass means.

Algorithm	**Top-1**	**Top-2**	**Top-3**	**Top-4**	**Top-5**
Baseline	69.47%	81.02%	86.15%	89.34%	91.48%
XBM	75.36%	84.92%	89.15%	91.65%	93.36%
XBN	75.37%	84.81%	88.94%	91.53%	93.35%

Encouragingly, across the various approaches, the embeddings tend to be relatively close to their superclass means, with over two-thirds being clustered most tightly to their ground truth superclass mean. As a result of their demonstrated ability to assist in increasing the separability of the embedding space, the XBM and XBN approaches show an improvement over the baseline in terms of superclass clustering. This result also empirically highlights the correlation between the semantic meaning of the data and its distribution in the embedding space. Overall, we see strong justification for using superclass normalisation, as the embeddings are already relatively close to their superclass means. We now evaluate the SC-XBN algorithm on all datasets with superclasses and find positive results. Excluding In-Shop, SC-XBN performs within 0.06 of the best of the previously explored approaches on Recall@1, with at least 0.19 *increase* in Recall@10. This is a significant result and a particular improvement over the PC-XBN approach. The results are shown in Table 5.17.

Table 5.17: Results of the SC-XBN algorithm on the CIFAR-100, In-Shop, SOP, and DeepFashion datasets.

	CIFAR-100	**In-Shop**	**SOP**
Algorithm	Recall@1 | Recall@10	Recall@1 | Recall@10	
Best (XBM, XBN, PC-XBN)	**82.01** | 91.68	**71.68** | **91.12**	**58.73** | 74.11
SC-XBN	81.95 | **91.87**	67.55 | 88.76	58.69 | **74.34**

The results are one of a few positive markers of SC-XBN. We see significantly more

stability when considering the embedding space than the PC-XBN approach, as shown in Figure 5.16. The drift of the SC-XBN algorithm is more stable than the PC-XBN approach, with a clear downward trend over the training period. A stable drift is a significant improvement over the PC-XBN approach, whose marked feature was erratic drift over the training period.

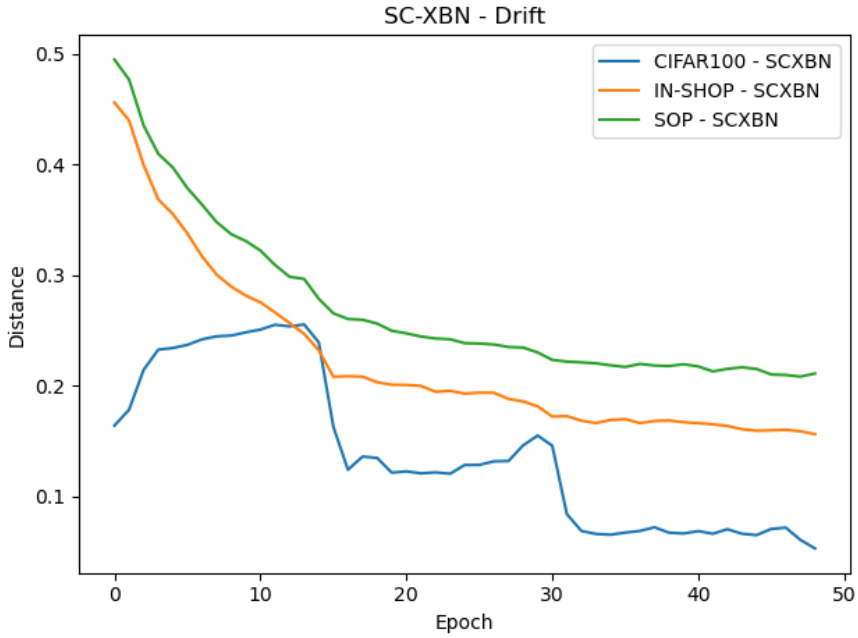

Figure 5.16: SC-XBN drift over time

The above positive results are not to say that SC-XBN in its current form is not without its issues. The model trained on In-Shop appears to succumb to a sudden collapse of the embedding space shortly after it reaches its peak performance (Fig. 8.22)whilst the model trained on SOP experiences decreasing performance past the tenth epoch (Fig. 8.23). DeepFashion2 proved incredibly slow to train

Overall, the SC-XBN algorithm demonstrates a significant improvement over the PC-XBN approach in the context of the CIFAR-100 dataset. This suggests that an increased granularity normalisation process can produce positive results when compared to traditional XBN.

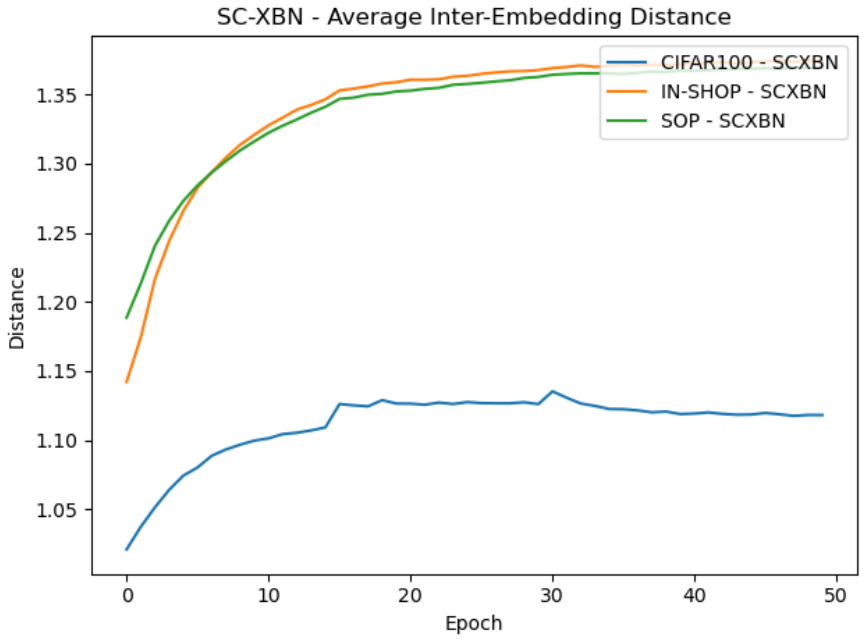

Figure 5.17: Average SC-XBN Inter-embedding distances

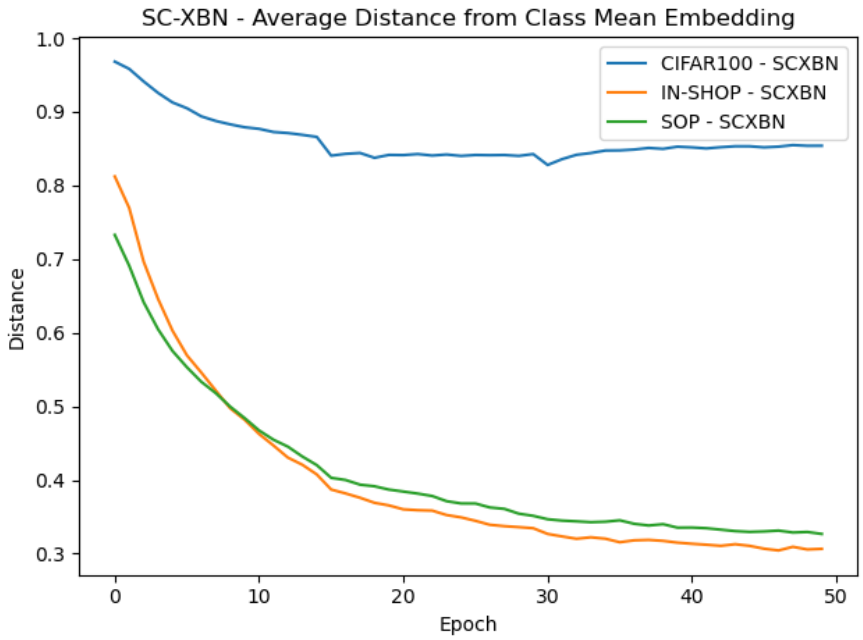

Figure 5.18: Average SC-XBN Intra-class distances

Chapter 6

Concluding Remarks

This thesis aimed to explore the intricacies of Cross-Batch Memory and its normalised variant (XBN) within the domain of deep metric learning. Driven by the desire to enhance the efficiency and adaptability of these techniques, we conducted a comprehensive evaluation of XBM and XBN, along with our proposed modifications and additional XBN variants, across a diverse set of benchmark datasets. Our investigation illuminated several vital findings that contribute valuable insights to the field of deep metric learning (DML).

Our experiments revealed that XBN, despite its normalisation mechanism, did not consistently outperform the standard XBM approach despite previous work suggesting the contrary. This indicates that the global normalisation introduced in XBN may only sometimes provide substantial benefits over the inherent adaptability of XBM, or at least, not in a robust out-of-the-box way. Furthermore, we identify that the standard deviation normalisation contibutes little to the normalisation process with centring showing similar, and in some cases superior, performance to XBN. However, we identified a crucial oversight in the original XBN implementation, where the Euclidean distance-based miner was not aligned with the normalisation process. Our findings underscore the importance of either unit-normalising XBN embeddings or employing cosine similarity in the miner to achieve optimal performance with XBN. This highlights how normalisation can affect various components of the DML pipeline and the importance of ensuring consistency across these components.

Furthermore, the proposed Delayed Normalisation modification, which postpones the normalisation of the memory bank, consistently yielded positive results across various datasets. This suggests that delaying normalisation allows the model to explore the embedding space more freely during the initial training stages, potentially preventing premature convergence to suboptimal solutions. This finding underscores the dynamic nature of the embedding space in DML and the importance of adapting the normalisation

strategy to accommodate this dynamism.

While the per-class normalisation modification, which performs normalisation independently for each class, promised to capture class-specific nuances in representational drift, it faced challenges due to its granular approach. The limited sample size per class, especially in datasets with a large number of classes, led to unstable statistics and hindered the effectiveness of this modification. The poor performance of PC-XBN highlights the trade-off between granularity and stability in normalisation techniques. Alternative approaches may be necessary for datasets with many classes or limited samples per class.

In contrast, the Superclass Normalisation modification, which leverages the statistics of broader superclasses, emerged as a promising compromise between the global normalisation of XBN and the granular per-class approach on the CIFAR-100 dataset. By operating at the superclass level, we used richer statistical information to improve performance. However, the applicability of this finding could have been more extensive and further applications of this approach are necessary to validate its effectiveness across a broader range of datasets. Despite this, Superclass Normalisation can effectively balance granularity and stability, offering a more robust normalisation strategy for specific cross-batch memory scenarios.

In conclusion, the exploration of XBM and its variants is this work has shed light on the nuances of these techniques and their impact on DML efficiency. Our proposed observations, particularly around delayed normalisation and Superclass Normalisation offer promising avenues for enhancing the adaptability and performance of XBM in various DML settings. By continuing to refine and explore these approaches, we can further advance the field of DML, paving the way for more efficient and robust metric learning models.

6.1 Future Work

The widespread use of cross-batch memory and the desire for more efficient deep metric learning models have led to significant research in this area. Whilst this work has deeply explored cross-batch memory and various modifications, there are still many avenues for future work.

6.1.1 Clustering

The PC-XBN and particularly SC-XBN results show similar performance to XBM and other methods in specific contexts. However, this work demonstrates three issues. Firstly, the level of appropriate granularity normalisation for a given task and dataset needs to be clarified. Secondly, the semantic distribution of the superclasses does not necessarily align with the distribution of the embeddings in the space learned by the model. Finally, superclasses are only sometimes available in the dataset. Hierarchical clustering is already performed for unsupervised tasks and unsupervised metric learning tasks, specifically (Yan et al., 2021), and could be used to identify the superclasses in the

embedding space for the PC-XBN/SC-XBN approach. Rather than using the class or superclass labels of the embeddings, the embeddings could be clustered after a warmup period or even continuously during training based on the movement of the embedding space. The embeddings would then be normalised based on the cluster they belong to rather than the class or superclass. Clustering would allow for a higher fidelity control over the granularity at which we perform normalisation, doesn't require the class or superclass labels to be known, and could improve model performance. Mainly related to the SC-XBN approach, using an appropriate clustering approach could overcome the disconnect between the semantic distribution of the superclasses and the actual distribution of the embeddings in the space learned by the model.

6.1.2 Alternative Normalisation Techniques

Whilst this work has shown that XBN does not significantly improve performance when compared to XBM and more complex normalisation shows diminishing returns, there are certainly more advanced normalisation techniques to be employed, which may enhance the performance of XBM. A straightforward approach would be to use a weighted average of the memory bank based on how recently the embeddings were added to the memory bank. This would allow the model to give more weight to more recent embeddings, which are more likely to represent the current state of the embedding space. Various similar approaches could be employed to improve the quality of the memory bank and, therefore, the model's performance.

A more advanced normalisation approach would be moving away from the mini-batch's first two moments. Instead, one could employ a strategy such as Riemannian Normalisation. Riemmanian averaging computes the mean of the embeddings on the manifold of the embeddings, which, in this work, is the hypersphere. Riemmanian averaging ensures that the normalisation results in embeddings more representative of the embedding space than the first two moments of the mini-batch. This approach has been shown to improve performance in deep learning when used for batch normalisation (Cho and Lee, 2017) and could be a potential avenue for future work. In particular, this approach would be particularly relevant for applications using Hyperbolic Metric Learning, where the embeddings would be normalised to the hyperboloid rather than the hypersphere (Yan et al., 2021).

6.1.3 XBN and Loss Functions

PC-XBN produced poor results, and we attribute this behaviour to the poor quality of the normalisation statistics. Centre contrastive loss (CCL) proposes a loss function that produces performance increases due to its movement of embeddings towards a class centre. CCL or a similar loss function with the PC-XBN approach could improve the quality of the normalisation statistics. The CCL loss function would move the embeddings towards the class centre, producing a mean that is more representative of the class and, therefore, improve the quality of the normalisation statistics. Exploring

more appropriate loss functions for normalisation could be a potential avenue for future work.

Chapter 7

Appendix: Full Results

Table 7.1: Full results across all experiments and In-Shop, SOP and DeepFashion2.

Algorithm	**In-Shop** Recall@1 | Recall@10	**SOP** Recall@1 | Recall@10	**DeepFashion2** Recall@1 | Recall@10
No-XBM	47.67 | 76.62	44.68 | 60.83	24.50 | 43.06
XBM	**71.68** | 91.12	58.73 | 74.11	43.63 | 62.57
XBN	71.02 | 90.95	58.63 | 73.78	43.53 | 61.33
PC-XBN	10.04 | 25.71	41.76 | 57.71	11.18 | 22.55
SC-XBN	67.55 | 88.76	58.69 | 74.34	-
XBN (15 epoch delay)	71.50 | -	58.90 | -	50.60 | -
Centre Normalised XBN	71.57 | -	**58.96** | -	43.5 | -
Scaling Normalised XBN	71.35 | -	58.64 | -	**49.66** | -

Table 7.2: Full results across all experiments and CIFAR datasets.

Algorithm	**CIFAR-10** Recall@1 | Recall@10	**CIFAR-100** Recall@1 | Recall@10
No-XBM	**96.95** | 98.98	77.24 | 91.11
XBM	96.85 | 98.86	**82.01** | 91.68
XBN	96.90 | 98.83	81.91 | 91.69
PC-XBN	85.69 | 96.67	39.69 | 77.36
SC-XBN	-	81.95 | 91.87
XBN (15 epoch delay)	96.90 | -	82.00 | -
Centre Normalised XBN	96.81 | -	81.97 | -
Scaling Normalised XBN	96.90 | -	81.7 | -

The best results are shown in bold. The SC-XBN algorithm was not evaluated on the DeepFashion2 dataset.

Chapter 8

Appendix: Experimental Graphs

8.1 No-XBM Baseline

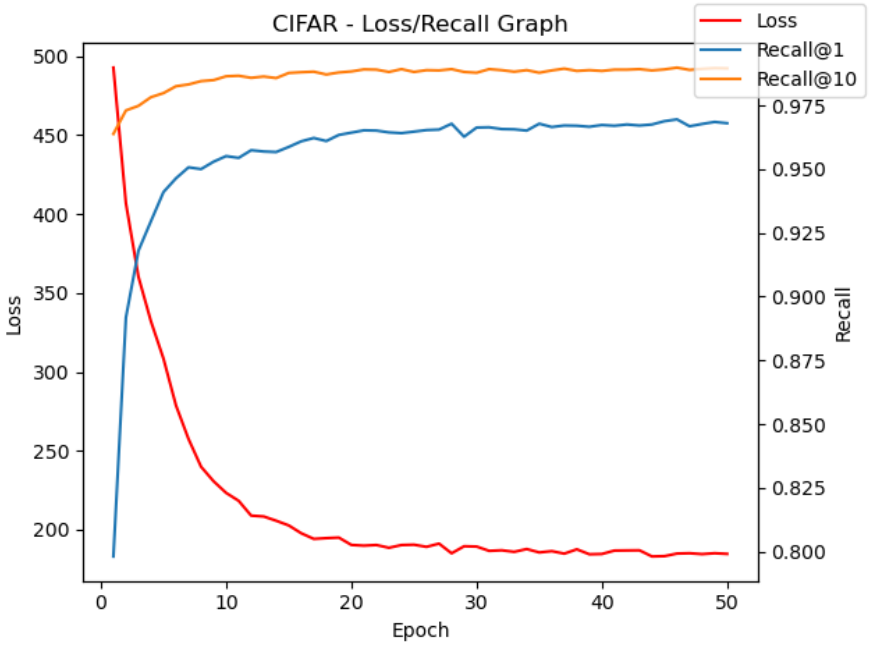

Figure 8.1: Cifar Training

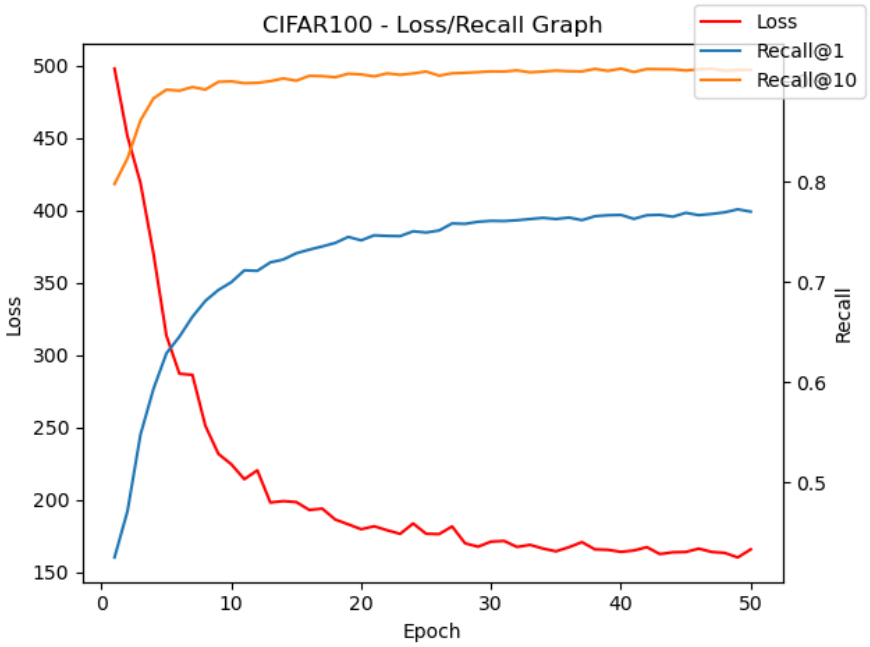

Figure 8.2: Average drift of class embeddings over training epochs

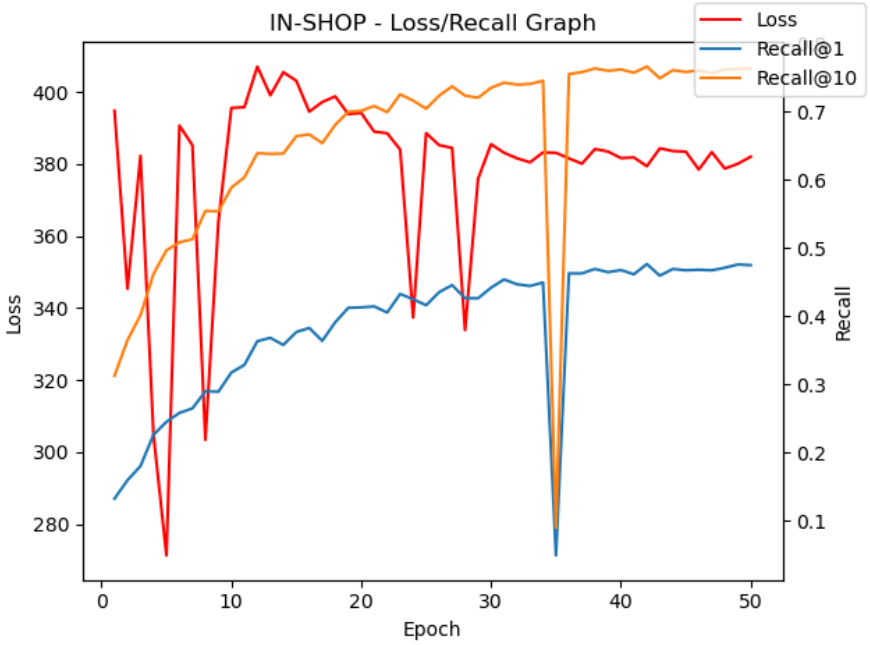

Figure 8.3: In-Shop Training

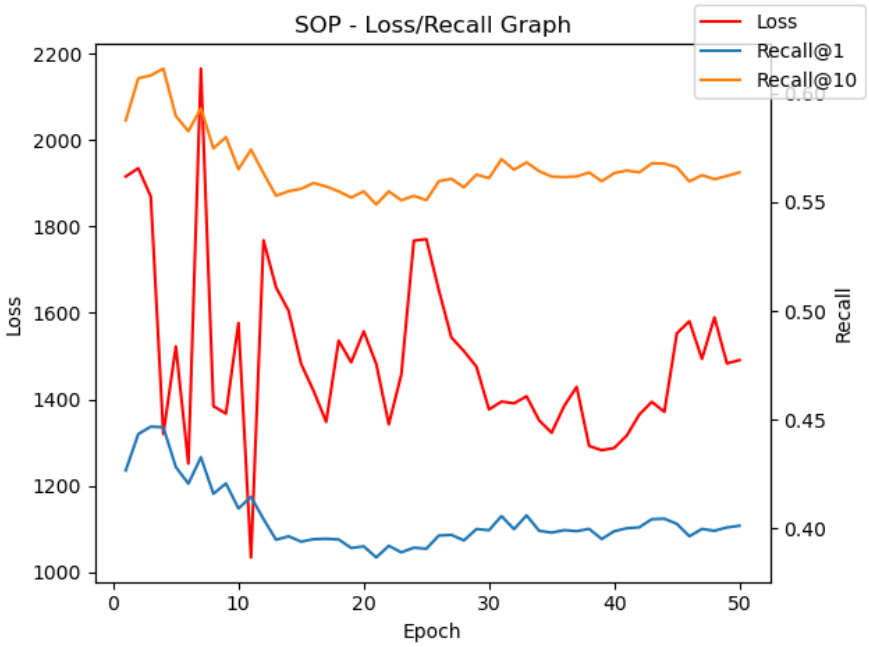

Figure 8.4: SOP Training

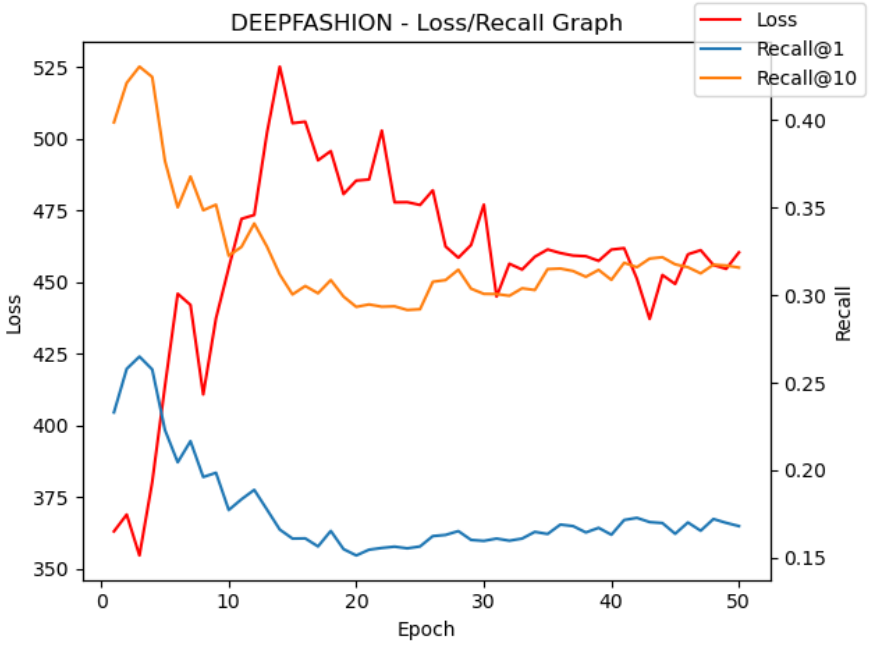

Figure 8.5: DeepFashion2 Training

8.2 XBM

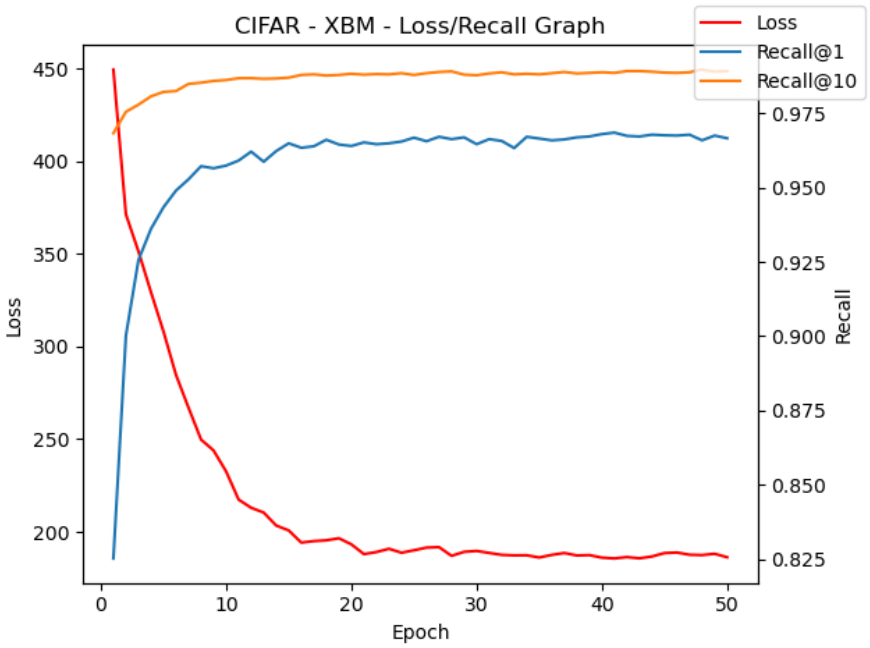

Figure 8.6: CIFAR XBM Training

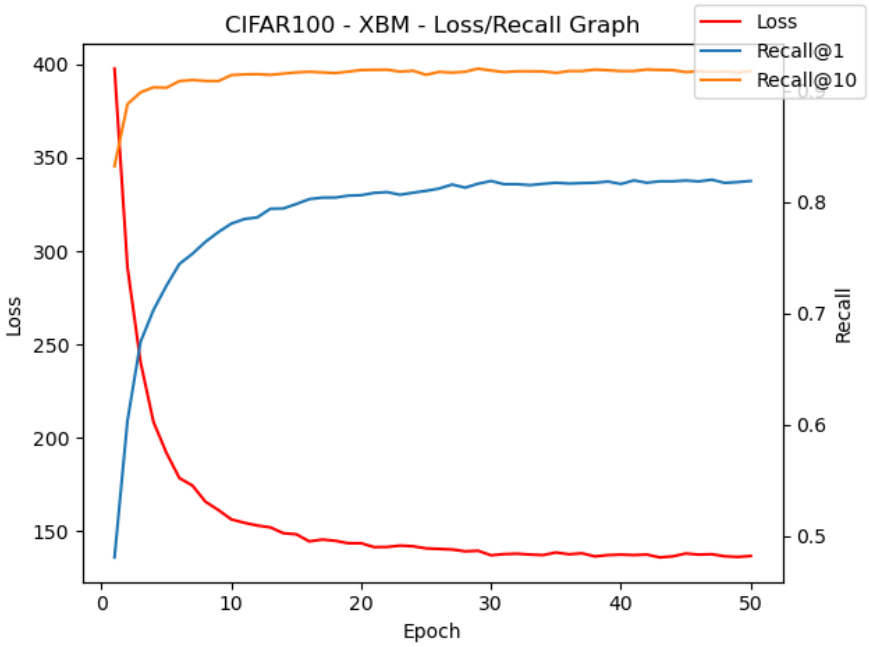

Figure 8.7: CIFAR-100 XBM Training

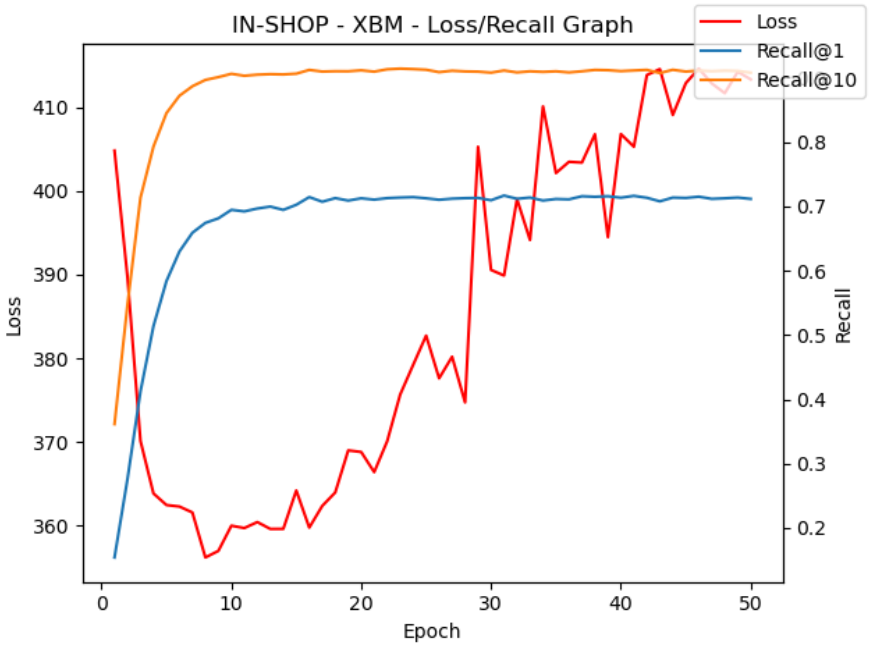

Figure 8.8: In-Shop XBM Training

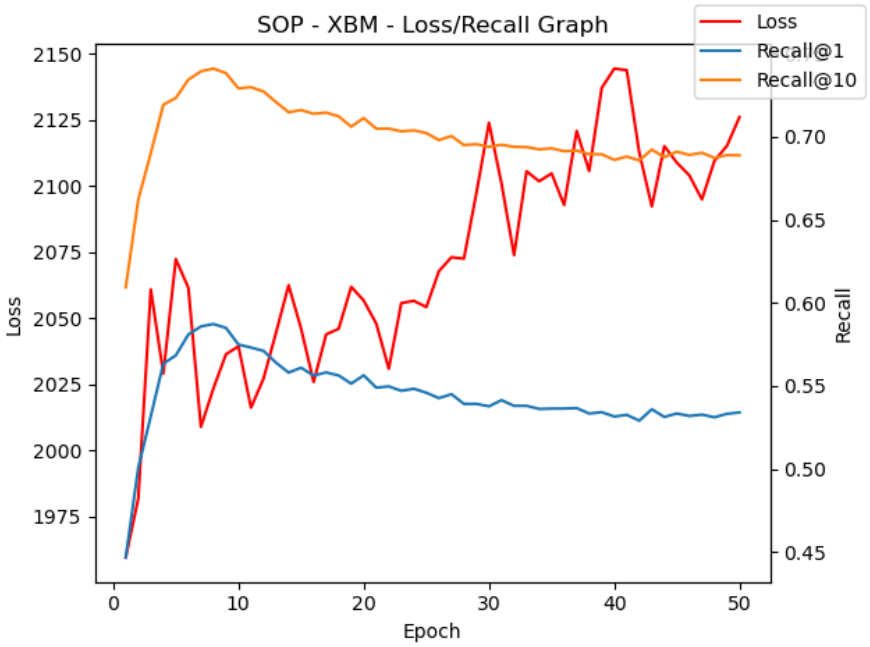

Figure 8.9: SOP XBM Training

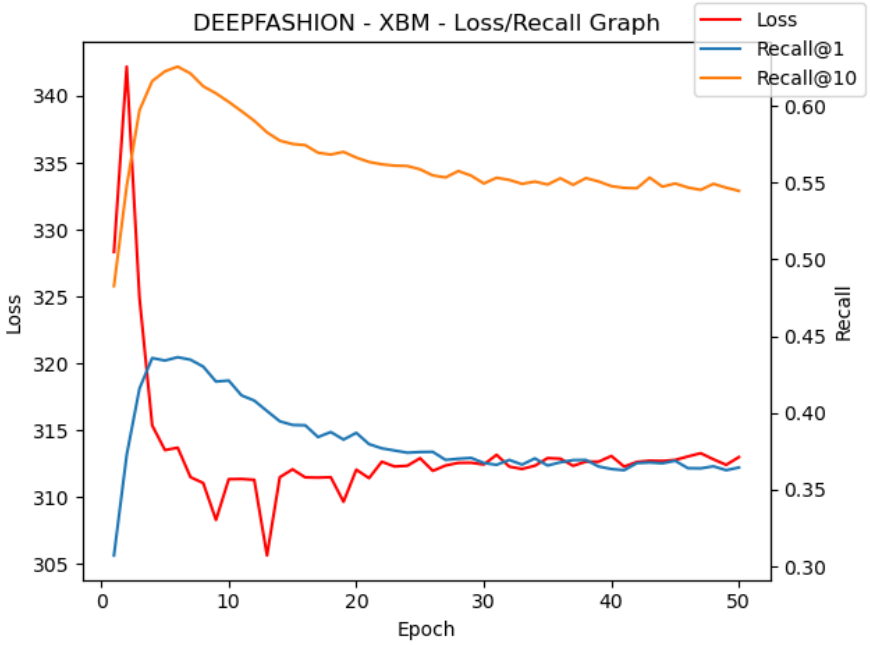

Figure 8.10: DeepFashion2 XBM Training

8.3 XBN

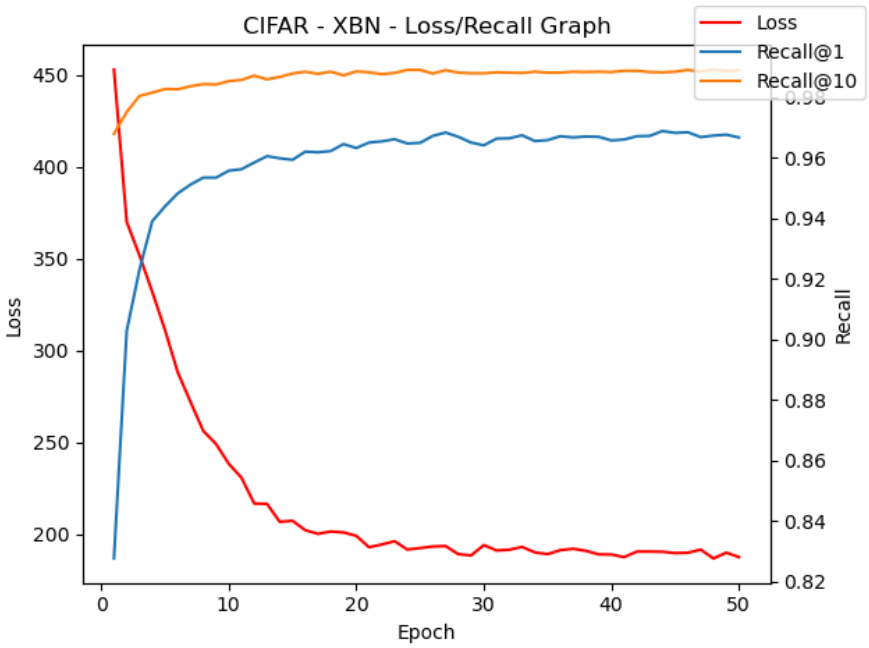

Figure 8.11: CIFAR XBN Training

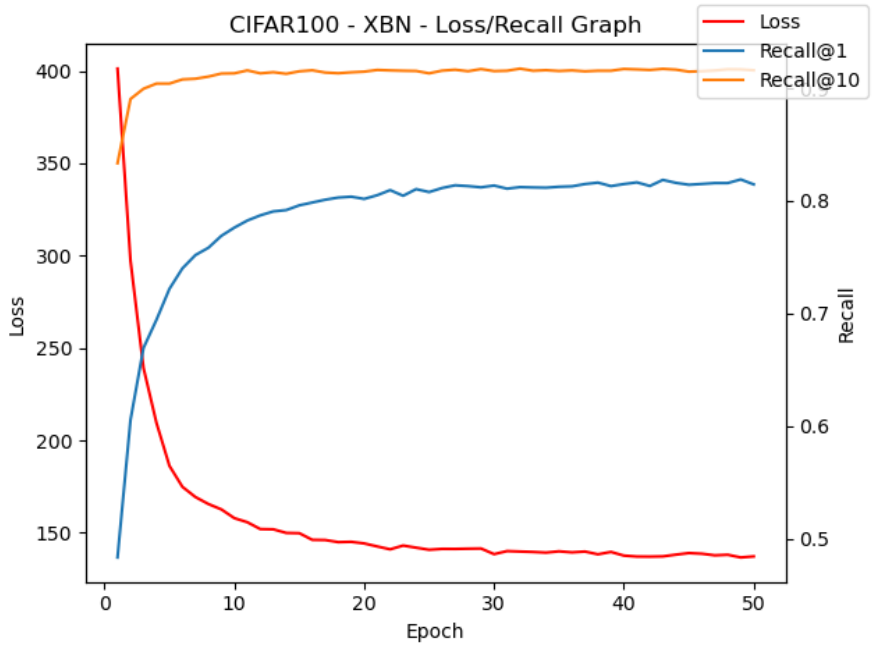

Figure 8.12: CIFAR-100 XBN Training

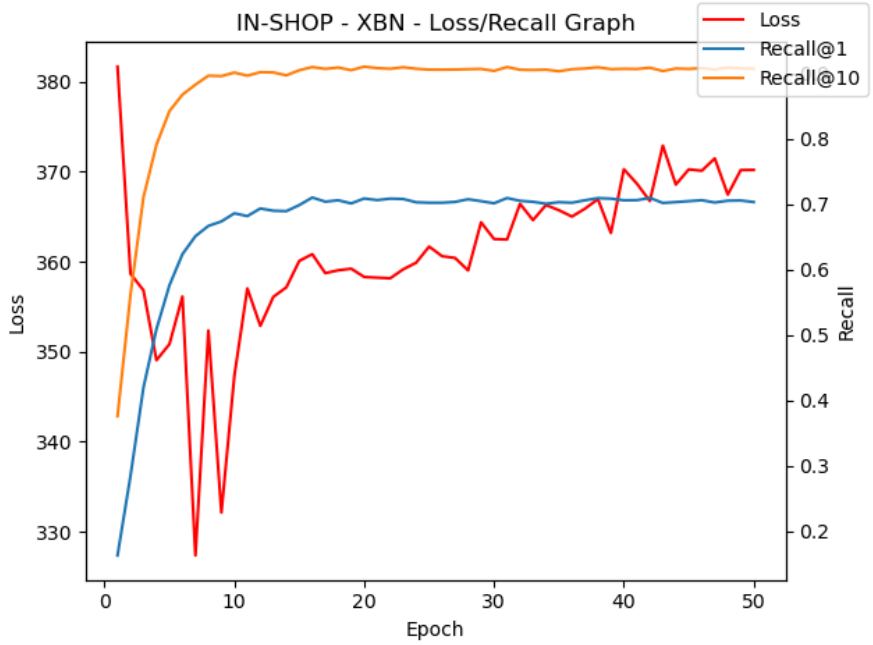

Figure 8.13: In-Shop XBN Training

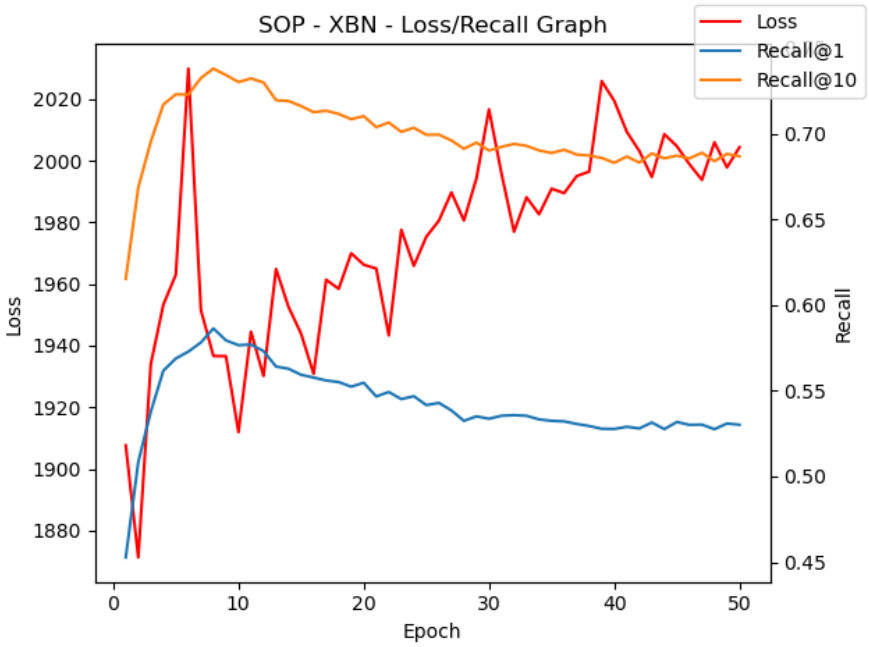

Figure 8.14: SOP XBN Training

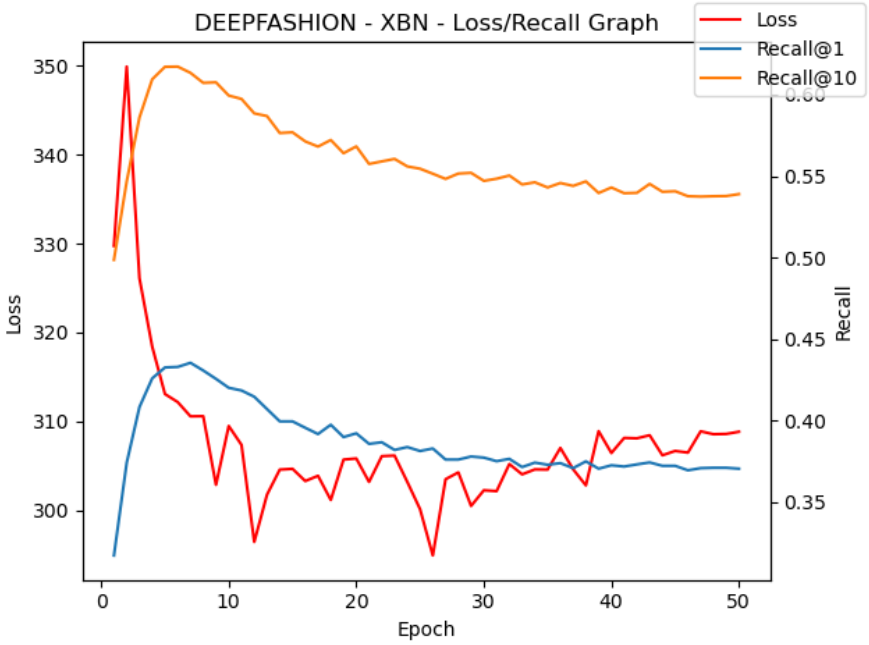

Figure 8.15: DeepFashion2 XBN Training

8.4 PC-XBN

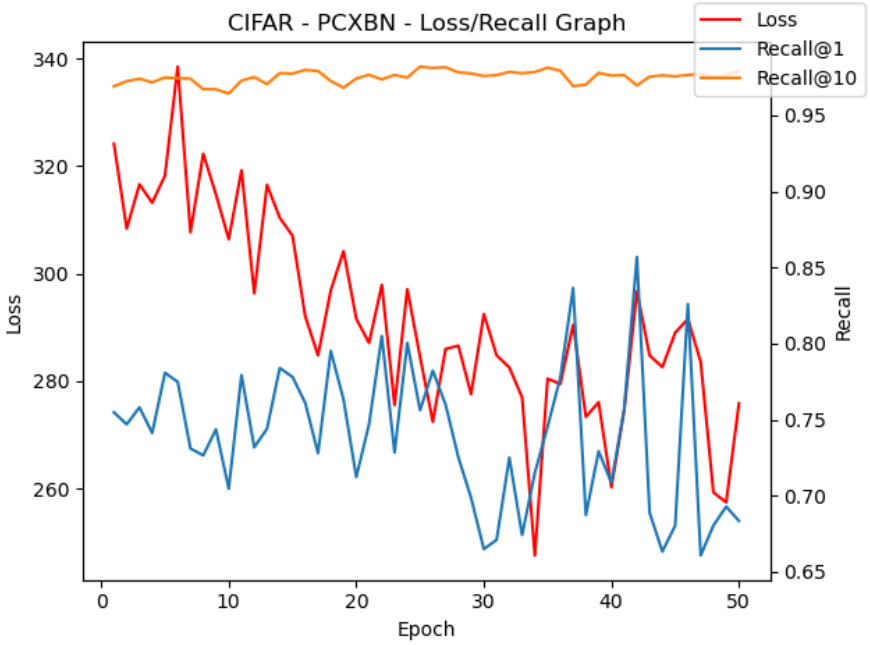

Figure 8.16: CIFAR PC-XBN Training

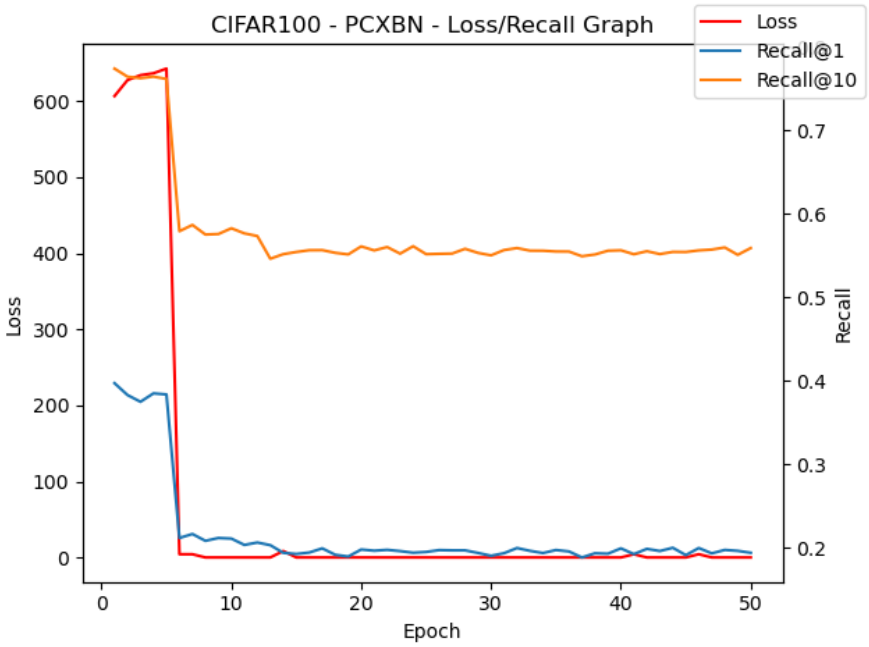

Figure 8.17: CIFAR-100 PC-XBN Training

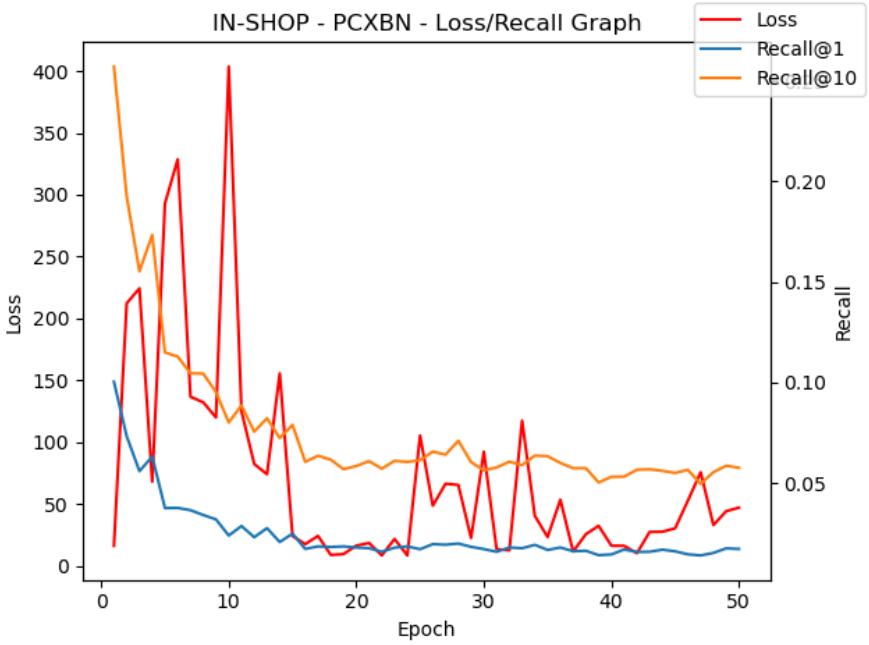

Figure 8.18: In-Shop PC-XBN Training

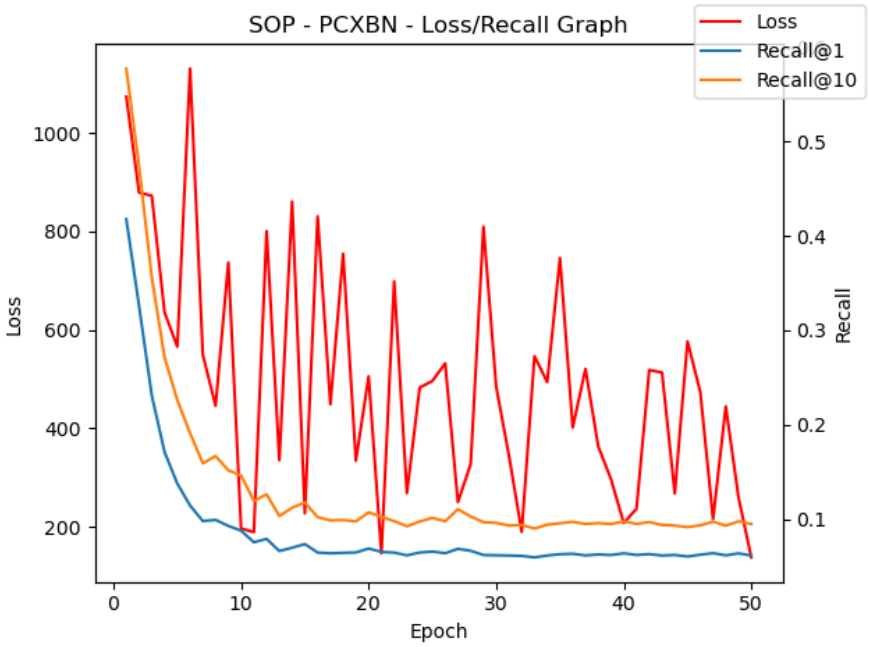

Figure 8.19: SOP PC-XBN Training

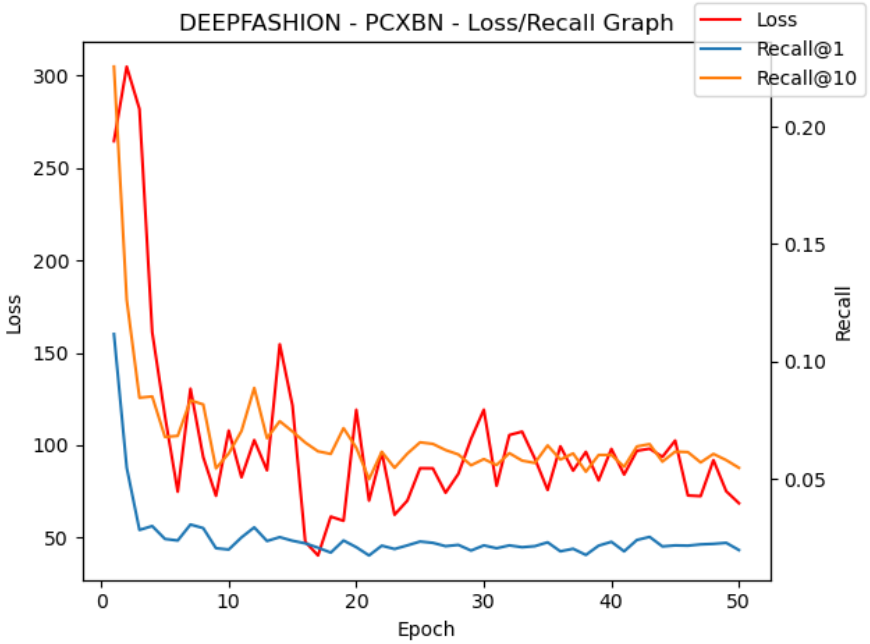

Figure 8.20: DeepFashion2 PC-XBN Training

8.5 SC-XBN

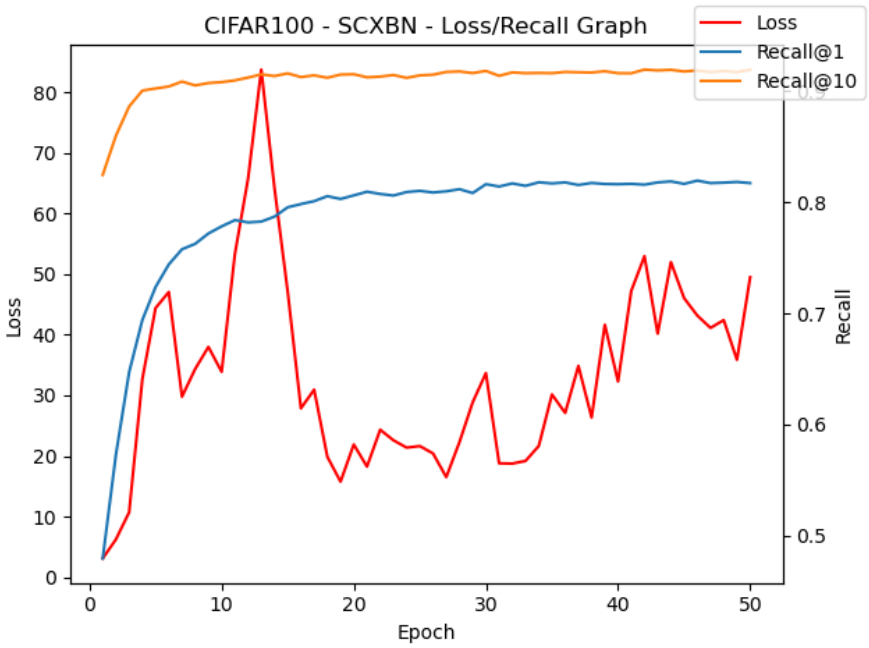

Figure 8.21: CIFAR-100 SC-XBN Training

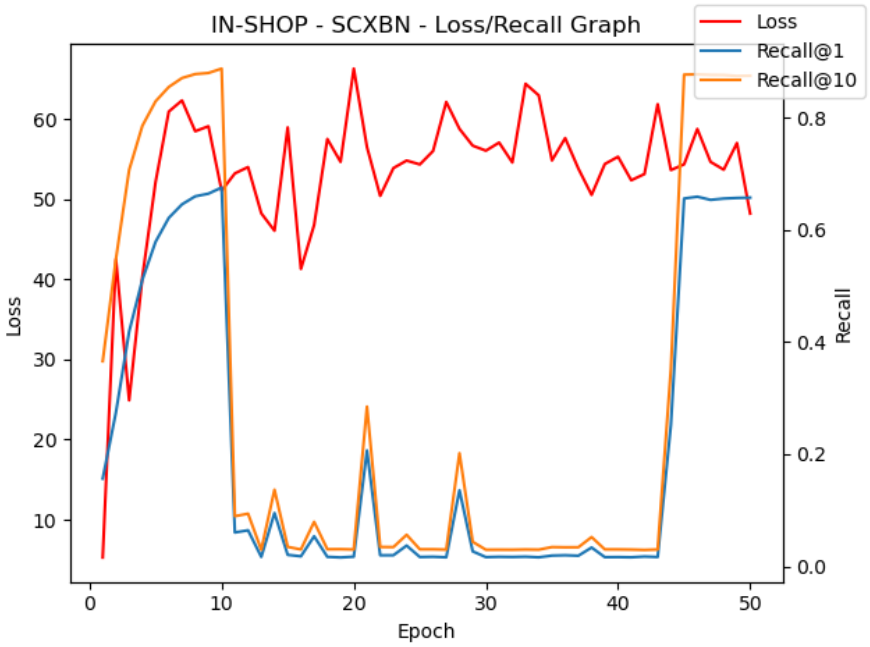

Figure 8.22: In-Shop SC-XBN Training

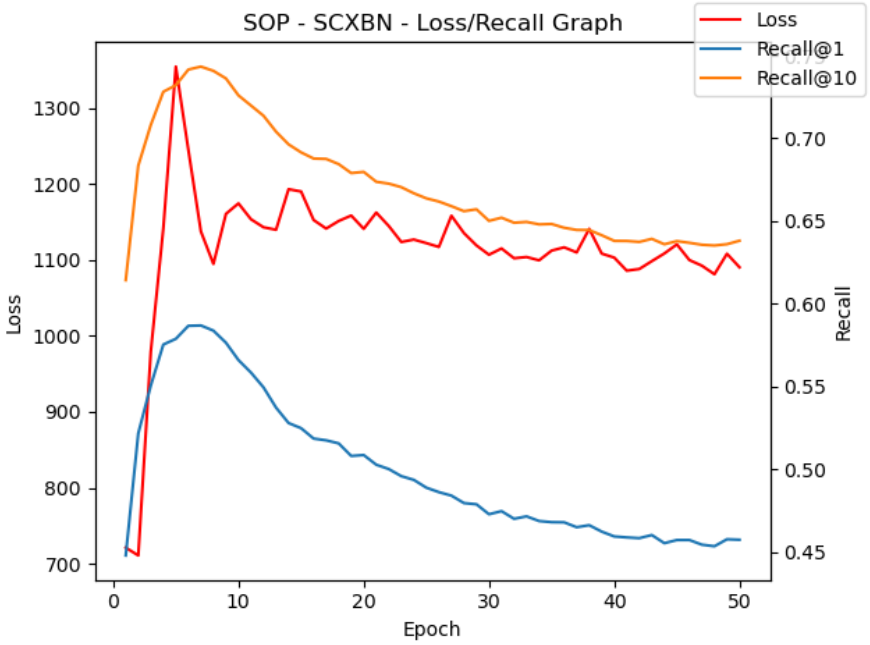

Figure 8.23: SOP SC-XBN Training

Chapter 9

Appendix: Non-Standard Experimental Parameters

9.1 PC-XBN (Margin Adjusted)

Table 9.1: PC-XBN Parameters

Parameter	Value
Positive Margin	0.1
Negative Margin	2.0
λ_μ	0.5
λ_δ	1

9.2 SC-XBN

Table 9.2: PC-XBN Parameters

Parameter	Value
Positive Margin	0.1
Negative Margin	2.0
λ_μ	0.5
λ_δ	1

Bibliography

Ajanthan, T.; Ma, M.; van den Hengel, A.; and Gould, S., 2023. Adaptive cross batch normalisation for metric learning. (Mar. 2023). [Cited on pages 2, 15, 17, 18, 20, 21, 23, 25, 26, 27, 28, 29, 37, 38, 40, 41, and 46.]

Cai, B.; Xiong, P.; and Tian, S., 2023. Center contrastive loss for metric learning. https://arxiv.org/abs/2308.00458. [Cited on page 14.]

Cho, M. and Lee, J., 2017. Riemannian approach to batch normalisation. https://arxiv.org/abs/1709.09603. [Cited on page 55.]

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L., 2009. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, (2009), 248–255. https://api.semanticscholar.org/CorpusID:57246310. [Cited on page 7.]

Ge, Y.; Zhang, R.; Wu, L.; Wang, X.; Tang, X.; and Luo, P., 2019. A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images. *CVPR*, (2019). [Cited on pages 14 and 26.]

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y., 2014. Generative adversarial networks. https://arxiv.org/abs/1406.2661. [Cited on page 7.]

Hadsell, R.; Chopra, S.; and LeCun, Y., 2006. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1735–1742. IEEE. [Cited on page 12.]

Harwood, B.; Kumar, V. K. B. G.; Carneiro, G.; Reid, I.; and Drummond, T., 2017. Smart mining for deep metric learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [Cited on pages 16, 17, and 30.]

He, K.; Zhang, X.; Ren, S.; and Sun, J., 2015. Deep residual learning for image recognition. (Dec. 2015). [Cited on pages 7, 8, and 9.]

Hertz, T., 2006. Learning distance functions: Algorithms and applications. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)* (Washington DC, 2006), 1–122. [Cited on page 5.]

Hoffer, E. and Ailon, N., 2014. Deep metric learning using triplet network. [Cited on page 13.]

Hornik, K.; Stinchcombe, M.; and White, H., 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 5 (1989), 359–366. doi: https://doi.org/10.1016/0893-6080(89)90020-8. https://www.sciencedirect.com/science/article/pii/0893608089900208. [Cited on page 7.]

Kalman, R. E., 1960. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82, 1 (03 1960), 35–45. doi:10.1115/1.3662552. https://doi.org/10.1115/1.3662552. [Cited on page 18.]

Kaplansky, I., 1977. *Set theory and metric spaces*. Chelsea Pub. Co., New York. [Cited on page 3.]

Khosla, P.; Teterwak, P.; Wang, C.; Sarna, A.; Tian, Y.; Isola, P.; Maschinot, A.; Liu, C.; and Krishnan, D., 2020a. Supervised contrastive learning. (Apr. 2020). [Cited on pages 13 and 14.]

Khosla, P.; Teterwak, P.; Wang, C.; Sarna, A.; Tian, Y.; Isola, P.; Maschinot, A.; Liu, C.; and Krishnan, D., 2020b. Supervised contrastive learning. [Cited on page 14.]

Krizhevsky, A., 2009. Learning multiple layers of features from tiny images. https://api.semanticscholar.org/CorpusID:18268744. [Cited on pages 22 and 26.]

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf. [Cited on page 7.]

Liu, Z.; Luo, P.; Qiu, S.; Wang, X.; and Tang, X., 2016. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [Cited on pages 14 and 25.]

Mohan, D. D.; Jawade, B.; Setlur, S.; and Govindaraj, V., 2023a. Deep metric learning for computer vision: A brief overview. [Cited on pages 1 and 7.]

Mohan, D. D.; Jawade, B.; Setlur, S.; and Govindaraj, V., 2023b. Deep metric learning for computer vision: A brief overview. (Dec. 2023). [Cited on page 11.]

Musgrave, K.; Belongie, S.; and Lim, S.-N., 2020a. A metric learning reality check. https://arxiv.org/abs/2003.08505. [Cited on pages 6, 12, and 15.]

Musgrave, K.; Belongie, S. J.; and Lim, S.-N., 2020b. Pytorch metric learning. *ArXiv*, abs/2008.09164 (2020). [Cited on page 26.]

O'Shea, K. and Nash, R., 2015. An introduction to convolutional neural networks. [Cited on page 7.]

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 6 (1958), 386–408. doi: 10.1037/h0042519. http://dx.doi.org/10.1037/h0042519. [Cited on page 6.]

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J., 1986. Learning representations by back-propagating errors. *Nature*, 323 (1986), 533–536. https://api.semanticscholar.org/CorpusID:205001834. [Cited on page 6.]

Schmidhuber, J., 2014. Deep learning in neural networks: An overview. (2014). doi: 10.1016/j.neunet.2014.09.003. [Cited on page 7.]

Schroff, F.; Kalenichenko, D.; and Philbin, J., 2015. Facenet: A unified embedding for face recognition and clustering. (2015). doi:10.1109/CVPR.2015.7298682. [Cited on pages 11, 12, 15, 16, and 47.]

Snell, J.; Swersky, K.; and Zemel, R., 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf. [Cited on pages 1 and 12.]

Sohn, K., 2016. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf. [Cited on page 14.]

Song, H. O.; Xiang, Y.; Jegelka, S.; and Savarese, S., 2016. Deep metric learning via lifted structured feature embedding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [Cited on pages 14 and 25.]

Wang, J.; Zhu, J.; and He, X., 2021. Cross-batch negative sampling for training two-tower recommenders. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21 (Virtual Event, Canada, 2021), 1632–1636. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3404835.3463032. https://doi.org/10.1145/3404835.3463032. [Cited on pages 17 and 23.]

Wang, X.; Zhang, H.; Huang, W.; and Scott, M. R., 2019. Cross-batch memory for embedding learning. [Cited on pages 1, 6, 16, 17, 19, 23, 25, 27, 30, 33, 40, and 43.]

Wightman, R., 2019. Pytorch image models. https://github.com/rwightman/pytorch-image-models. doi:10.5281/zenodo.4414861. [Cited on page 26.]

Wightman, R.; Touvron, H.; and Jégou, H., 2021. Resnet strikes back: An improved training procedure in timm. [Cited on pages 22 and 26.]

Yan, J.; Luo, L.; Deng, C.; and Huang, H., 2021. Unsupervised hyperbolic metric learning. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 12460–12469. doi:10.1109/CVPR46437.2021.01228. [Cited on pages 54 and 55.]

Zhang, Z.; Kwok, J. T.; and Yeung, D.-Y., 2003. Parametric distance metric learning with label information*. *IJCAI*, (2003). [Cited on page 5.]