A Geometric Approach to Image Classification using Neural Networks with Class Embeddings

Nikolai Aaen Bonderup, Henrik Nørgaard Ginnerup, Christian Bøgh Larsen,
Sebastian Lassen, Simon Brun Olsen, Steven Tran
{nbonde, hginne, cbla, slasse, solse, stran}19@student.aau.dk

Department of Computer Science, Aalborg University

***Abstract*—This paper investigates an interpretation of the classical neural network structure that elicits geometric meaning from input images. A new model, the Moving Targets Model, is defined, and reuses parts of neural networks but with the addition of class embeddings. Class embeddings are new vectors of learned values that represent classes. Five new loss functions are defined for the model. The loss functions are designed with the geometric meaning of the input images in mind. To observe the behaviour of the proposed model, the task of image classification is selected. Variations of neural networks are implemented and tested against popular image classification tasks. Since standard classifiers do not handle class embeddings, methods capable of using class embeddings are designed and used during experiments. Experiments show that the Moving Targets Model achieves similar results to baseline models, while confirming the successful application of loss functions and providing a geometric meaning of images. Further development of the Moving Targets Model and loss functions that better exploit the geometric meaning of images is left unexplored. This project is open source[1].**

***Index Terms*—Class embedding, Image classification, Neural network, Deep learning, Embeddings, Loss functions**

I. Introduction

An almost ubiquitous characteristic of all neural network models used for image classification is the final layer being a fully connected one, mapping the previous layers to the output classes [40]. Such networks typically function by having the early layers learn more general features of the input images such as corners and textures. The later layers then combine the outputs of these earlier layers to create higher-level representations, which are typically more specific to the classes [11]. The fully connected last layer handles the task of classification based on the extracted features. This seems perfectly reasonable, and probably is, as convolutional neural networks were heavily utilised by state-of-the-art models until the recent introduction of vision transformers [3]. But could anything be gained from not doing this and instead directly examining and using the d-dimensional embeddings found between the penultimate layer and the classifier?

In neural network models, embeddings are the data representations between each layer. As a neural network takes in an image at one end and outputs a classification at the other end, the embeddings throughout the network can be seen as a gradual transformation of the data from the image to the classification. In the use of neural networks for classification, embeddings provide a window into the inner workings of the network. The embedding right before the last layer is often used for this purpose. From the notion of embeddings being gradual transformations throughout the network, the embedding before the last layer represents all the knowledge extracted from the image that is needed for classification.

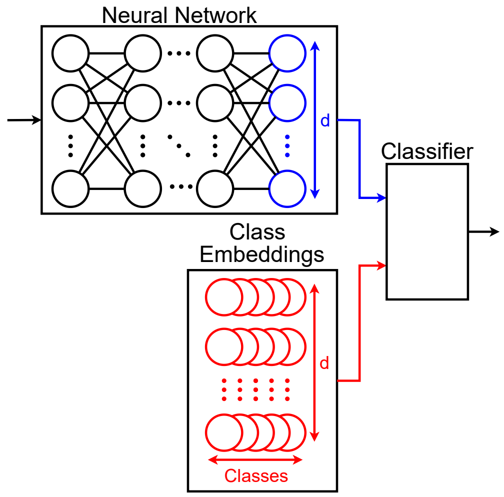


Fig. 1: The Moving Targets Model consisting of a neural network that outputs embeddings, class embeddings and a classifier.

In this paper, models for image classification which directly use this embedding as the output of a neural network are constructed and examined. The embeddings are constructed by interpreting the last layer of a neural network separately from a classifier. Thereby, the modified neural network simply translates the classification problem into a simpler one, rather than providing the classification itself. This means that both the network and the, now separate, classifier can be changed somewhat independently. A high level overview of the model is presented in Figure 1. Class embeddings are d-dimensional representations of the classes as vectors which are also learnt during training. The merits and effectiveness of this approach is investigated, along with any insights into classification using neural networks it may provide. For this approach to function, alternative methods of classification, such as variants of nearest neighbour methods, need to be designed and evaluated. Additionally, different methods for calculating the loss in such models are designed and how they compare to common, linear layer-based classification methods is examined.

The domain of image classification is chosen as the proving ground for this approach. The task of image classification has become increasingly important and much development has gone into improving image classification methods to be more accurate, less data-intensive, and overall faster [47]. The

[1]https://github.com/Simonbolsen/P8

successful use of convolutional neural networks (CNN) has especially pushed the development of image classification methods towards deep learning, and many of the most used image classification networks such as ResNet and AlexNet use tens, sometimes hundreds, of trainable layers [24][33].

This paper proposes a neural network model that provides a geometric representation of embeddings. It also defines several loss functions that utilise the chosen geometric representation of images, within the typically hidden embedding layer. Additionally, the paper describes several variations of nearest neighbour classifiers that work for the model and rigorously tests variations of the proposed methods. Finally, it compares them to a described baseline on several popular image classification tasks.

II. Related Work

The proposed method is designed for the task of classification, which is generally approached in two ways: supervised and unsupervised. For supervised image classification, a training-set with labelled images is required. For unsupervised, there is no labelled data and the task is often associated with clustering and pattern recognition. Several methods of varying complexity exist in image classification. Examples of these methods include decision tree classifiers [41][39], support vector machines [26][30], and neural network models [44][27]. The proposed methods is supervised and uses deep neural networks with convolutional layers and manipulation of a latent feature space (embeddings).

For deep neural networks used in image classification, the spatial information of an image is represented as vectors in a latent space called the embedding space. While the embedding space may be represented simply as a hidden layer within a neural network model [48][31][21][29], it may also be actively manipulated to achieve some desired effect. Arif et al. [47] manipulate the embedding space using ”phantom embeddings” which are points within the embedding space based on samples from the classes. The intention of the phantom embeddings is to decrease inter-class similarity by having a pulling mechanism from the embedded images towards their respective phantom embedding.

Barz et al. [9] seek to find a semantic representation of images using hierarchical information about image classes. Further, they transform images into an information rich latent embedding space, such that the inter-relational properties between image representations and their respective classes can be derived from their position within this space. They show that in such an embedding space, using negated dot product as similarity measure, an increase in classification accuracy can be achieved.

Sun et al. [43] present a label embedding network, where each label is mapped to a latent embedding space. They define this space such that each element of these embedding vectors for the labels provide a similarity measure between two labels. They note the adversity with increasing the number of labels and provide a compressed version of the label embedding network, which achieves a significant reduction in error compared to other methods.

Dwibedi et al. [18] focus on learning an encoder which maps images to some embedding space. Within this embedding space, Dwibedi et al. uses a nearest neighbour approach to find similar images within a support-set to learn a label for the input image. Since generalisation of models is an important factor for not overfitting on training data, models are encouraged to be robust when handling variations of images. Typically, variations may be produced by data augmentations such as rotating, cropping, and changing the colours of the images to produce several variations of an image. Dwibedi et al. instead use other images already seen by the model as opposed to generating new images via data augmentation.

Within manifold learning, where one tries to learn a mapping from high dimensional data to a lower dimensional embedding, the manipulation of embeddings and an embedding space is central as also seen from Zang et al. [51]. Zang et al. focus on employing neural networks, as opposed to predefined distance metrics, in an attempt to maintain the structural relationships in the data. Zang et al. manage to outperform other state-of-the-art methods within popular manifold learning tasks which coincide with popular image classification tasks.

The use of pre-trained models for image classification has been explored by Dosovitskiy et al. [17]. Dosovitskiy et al. examine the use of vision transformers for image classification. Transformers are used by state-of-the-art models within natural language processing [49], but Dosovitskiy et al. notes that transformers are not used for state-of-the-art methods within computer vision. They employ vision transformers for image classification and find that vision transformers perform well when pre-trained with a large dataset and then employed on data with fewer, distinct classes.

Typically, embeddings of images are kept hidden within deep neural network models as just another hidden layer [48][31][21][29], but others investigate and manipulate the embeddings and the embedding spaces to observe their effects [47][9][43][18]. Similarly, this article investigates the effect of actively moving the embeddings of the target classes to learn what effects this has on the network.

While this article concerns many of the same ideas and topics as the mentioned related works, it differs by adding so-called class embeddings, which are representations for each class within an embedding space. Similar representations are used by Arif et al. [47], but for this article, the placement of such representations are learned rather than calculated. This provides a geometric meaning to loss functions, as these will change the way each representation is moved in the latent space.

III. Moving Targets Model

The model proposed in this paper is referred to as the Moving Targets Model. The Moving Targets Model consists of three parts, as shown in Figure 1: a neural network, a classifier, and an embedding for each class referred to as a **class embedding**.

The neural network part of the Moving Targets Model is based on common image classification architectures with slight modifications. A common image classification architecture consists of a set of convolutions followed by a set of linear layers. The final layer is commonly used for classification [24][33]. In the Moving Targets Model, the final layer is not used for classification and is instead removed. Class embeddings are added to the model in parallel with the neural network. As the last linear layer of the neural network is removed, the network now outputs a d dimensional vector. This vector is referred to as an **output embedding** and is marked in blue in Figure 1.

Class embeddings, marked in red in Figure 1, exist in the same vector space as output embeddings. Each class embedding in the model is associated with a single class, hence the name. The class embeddings are learned parameters, similar to weights and biases in a neural network. As such, they are not calculated based on the input to the model.

To perform predictions, a classifier is utilised as shown in Figure 1. The classifier is given an output embedding for a given image and all of the class embeddings. Different classifiers can be used here. Nearest neighbour classifiers are used in this paper due to their compatibility with class embeddings.

A. *Model purpose*

The purpose of the Moving Targets Model is to provide a geometric understanding of the embedding layer. As the embeddings can be interpreted to be within an embedding space, wherein images are mapped to points, a geometric understanding of such a space means that the points which images are mapped to should contain semantic information and meaning in relation to other points. One such geometric property that is investigated is grouping images belonging to the same class closer together, thus making classification based on distance metrics possible.

B. *Embedding space characterisation*

A consequence of the model is that the commonly used loss functions, such as mean squared error and cross-entropy [12], are no longer applicable in their usual forms. As such, other loss functions are devised. The following section provides an outline of the concepts used to characterise the embedding space. This is followed by a description of the loss functions used throughout the rest of the paper.

Firstly, the set of **class centres** G is introduced as the average position of all output embeddings belonging to a class. Class centres should not be confused with class embeddings, that are trained as a part of the model. Class centres are independent of the model and can thus be found for neural networks not trained with class embeddings. Thereby, class centres enable comparison of the output embeddings of the Moving Targets Model and the embeddings from a conventional neural network. Figure 2 shows output embeddings and class centres in an embedding space divided by which class centre is closest.

The **global centre** $avg(G) = \frac{1}{|G|} \sum_{g \in G} g$ is the point which is an equal distance from each class centre. The accuracy of the approximation is related to the symmetry of the positions of the class centres. The global centre is also the average output embedding $avg(G) = avg(E)$, under the assumption that there is an equal amount of images and thereby output embeddings of each class.

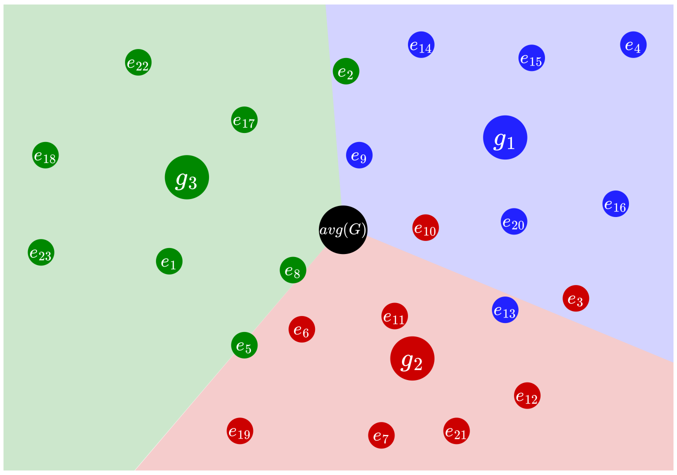


Fig. 2: A visualisation of an embedding space in two dimensions. The output embeddings e_i each belong to a class denoted by the colour. The entire space is also coloured by which **class centre** is the closest. Each class centre $g_j \in G$ represents the mean position of output embeddings that belong to class j.

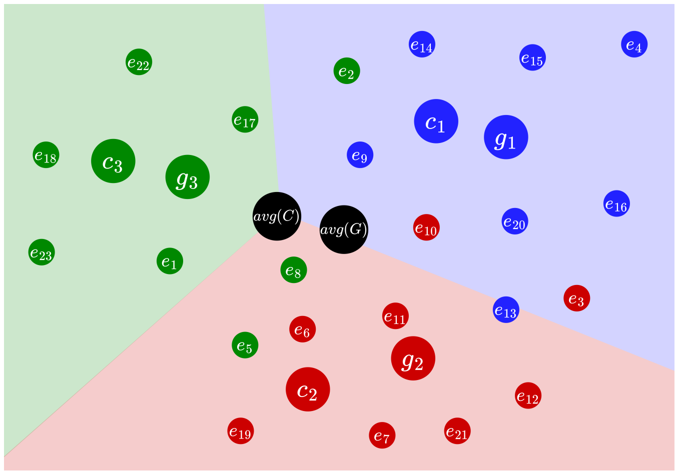


Fig. 3: A visualisation of an embedding space in two dimensions. The output embeddings e_i each belong to a class denoted by the colour. The entire space is also coloured by which **class embedding** is the closest. Each class centre $g_j \in G$ represents the mean position of output embeddings that belong to class j. Each class embedding $c_j \in C$ is the learned representation for class j.

Figure 3 shows a similar embedding space to Figure 2 but divided by class embeddings. Depending on the loss function, class embeddings may approximate class centres, but could for instance also be used to position output embeddings closer or further from the global centre $avg(G)$. Depending on the loss function, the average class embedding may serve as an approximation of the global centre to reduce the computational load, as class embeddings are learned values and class centres must be aggregated from all output embeddings. Loss functions which minimise the distance between output embeddings and class embeddings preserve that class embeddings approximate class centres.

A **cluster** is defined as all output embeddings belonging to the same class. The **cluster size** σ is here defined as the median distance of all points in the cluster to the class centre. This is thus the distance from the class embedding or class centre which half of the output embeddings of the class are

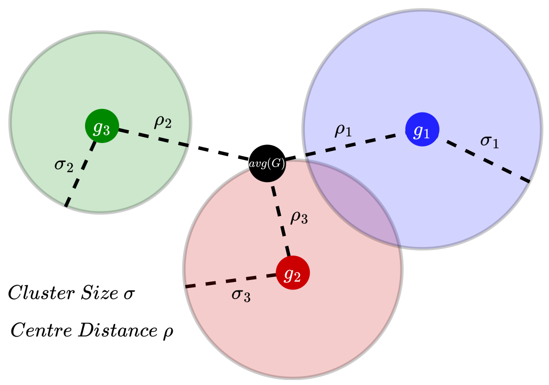


Fig. 4: The class clusters around class centres visualised with cluster sizes and centre distances. Note that as the cluster sizes denote the median distances only half of the output embeddings are within the circles shown.

within. This metric is chosen as clusters do not have clear boundaries and some output embeddings may be very far from the centre, which could skew a mean value.

In order for the nearest neighbour classifier to properly work, the size of the cluster must be smaller than the distance between the class centre or class embedding to the global centre. This distance between the class centre or class embedding to the global centre is referred to as the **centre distance** ρ. The assumption behind these measures is, as shown in Figure 4, that the greater the cluster size is, compared to the centre distance, the more the clusters overlap, and the harder it is to accurately classify them.

Cluster size and class centre both relate to a single class. Later these measures are utilised for all classes to compare variations of the model. When later referring to the measures they are therefore aggregated over all classes. The cluster size is aggregated over classes by taking the median of the distances from all output embeddings to their class centres rather than only including the output embeddings from a single class. The centre distances provide one distance for each class and are aggregated by an average.

C. Loss functions

This section outlines the loss functions used for the Moving Targets Model. These functions use the symbols shown in Table 1. Note that loss functions used for the model, in addition to the output of the neural network, also consider the class embedding for the class of the output. In the training process, the correct class is known for the input and thus the correct class embedding is also known.

Symbol	Description
e	The embedding output by the neural network
c_e	The class embedding for the class of e
E	The set of output embeddings
C	The set of class embeddings

Table 1: The variables used in loss functions.

If v and u are vectors, $|v|$ denotes the euclidean length of a vector such that $|v - u|$ is the euclidean distance between the vectors. $v \bullet u$ denotes the dot product of v and u.

It should be noted that both the output embeddings and class embeddings are updated through learning. Also note that the subsequent loss functions all express the loss for one instance; however, the actual implementation uses batches instead. For each batch in the training process, the loss used is thus the sum of the loss of all output embeddings in that batch.

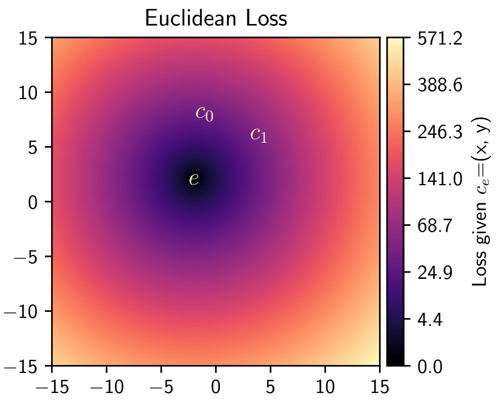


Fig. 5: $loss_{euclid}$ as a function of c_e given e and $C \setminus \{c_e\}$ in a 2-dimensional space.

1) *Euclidean Distance loss:*

The simplest geometric property that may allow for classification would be to cluster output embeddings of the same class closer together. The class embedding can then denote the centre of the cluster and thus be a representation of the class in the embedding space. One way to achieve this is to penalise the distance between output embeddings and their target class embeddings.

The Euclidean Distance loss function directly uses the squared euclidean distance to calculate the loss given output embedding e and its class embedding c_e. If the class embedding is interpreted as the target, the expression of this loss function differs from mean squared error in two ways. First, the class embeddings are changed through backpropagation, and secondly, the loss function is not normalised as a mean. The expression of the loss function can be seen in Equation (1).

$$loss_{euclid}(e, c_e) = |e - c_e|^2 \qquad (1)$$

With $loss_{euclid}$, the class embeddings will move to the centre of the cluster of output embeddings belonging to that class. There are however no guarantees as to whether the clusters of output embeddings of different classes overlap. If this is the case, nearest-neighbour classification does not work optimally.

Given an output embedding e, its target class embedding c_e, and two other class embeddings c_0 and c_1 different from c_e, Figure 5 illustrates the loss function as a function of c_e in a two-dimensional embedding space. Dark shades indicate a low loss while bright shades indicate a higher loss. As can be seen, having c_e closer to e results in a lower loss. However, even if c_e is close to c_0 or c_1, which would not be ideal when using nearest neighbour classification, the loss does not change.

2) *Cosine Distance loss:*

An alternative to the Euclidean Distance loss function is a loss function based on cosine similarity of two non-zero n-dimensional vectors [23].

$$cos(e, c_e) = \frac{e \bullet c_e}{|e| \cdot |c_e|} \qquad (2)$$

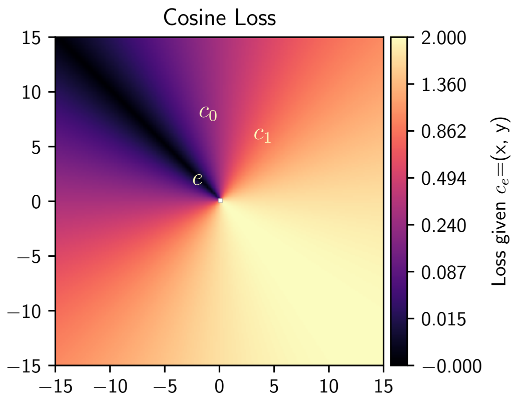


Fig. 6: $loss_{cos}$ as a function of c_e given e and $C \setminus \{c_e\}$. Note that vectors have their origin in $(0, 0)$ which causes the direction of high and low regions.

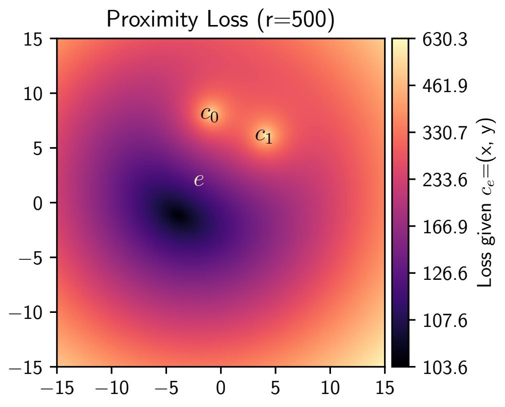


Fig. 7: $loss_{prox}$ as a function of c_e given e, $C \setminus \{c_e\}$, and $r = 500$. The loss decreases as c_e approaches e, but increases rapidly as c_e approaches any other $c_i \in C$.

The formula for cosine similarity can be seen in Equation (2) where e is an output embedding and c_e is a class embedding. The function outputs a number in the interval $[-1, 1]$ with -1 being opposite vectors, 0 meaning orthogonality, and 1 being parallel vectors. Cosine similarity can be used as a loss function as seen in Equation (3). This is also known as the Cosine Distance loss function.

$$loss_{cos}(e, c_e) = 1 - cos(e, c_e) \tag{3}$$

Figure 6 shows how the loss changes based on the position of target embedding c_e. The loss decreases as the angle between e and c_e gets smaller, as measured from the origin. When the angle between e and c_e is maximised, that is $cos(e, c_e) = (-1)$, the loss peaks.

3) Class Proximity loss: A possible weakness of the Euclidean Distance loss function is that nothing prevents the moving targets from converging to a single point. This section introduces the Class Proximity loss function, or Proximity loss for short. Proximity loss extends the Euclidean Distance loss function with a proximity measure loss penalty when classes are moved too close to each other. The proximity measure can be expressed as in Equation (4) where c_e is the target class embedding and C is the set of all class embeddings.

$$prox(c_e) = \sum_{k \in C \setminus \{c_e\}} \frac{1}{|c_e - k|^2 + \epsilon} \tag{4}$$

The inverse distances from c_e to any other class $c \in C$ is summed, and ϵ is a small number added in the denominator to prevent division by 0. The Class Proximity loss function can be seen in Equation (5).

$$loss_{prox}(e, c_e) = loss_{euclid}(e, c_e) + r \cdot prox(c_e) \tag{5}$$

In Equation (5), $r \in \mathbb{R}$ is added as a hyperparameter and determines the magnitude of the proximity penalty.

Figure 7 illustrates how the loss changes for Proximity loss as a function of the class embedding c_e in the embedding space and shows how there is a spike in the loss as c_e moves closer to either c_0 or c_1.

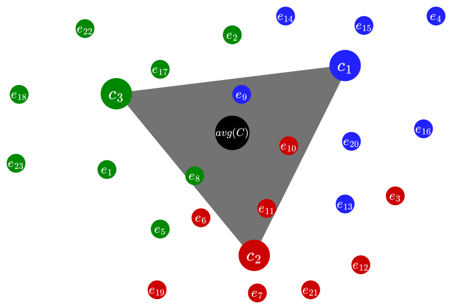


Fig. 8: The space between class embeddings for which classification may be impaired.

4) Push and Pull loss: One heuristic that may improve accuracy is to keep output embeddings out of the space between the class embeddings shown in Figure 8. To implement this in a loss function, a vector for the average class embedding can be computed as seen in Equation (6).

$$avg(C) = \frac{1}{|C|} \sum_{k \in C} k \tag{6}$$

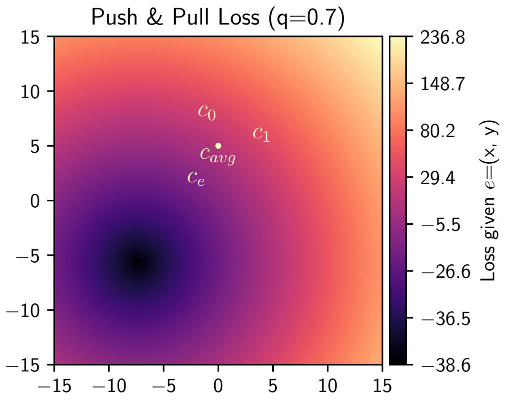


Fig. 9: $loss_{pnp}$ as a function of e given C. The loss function pulls e towards c_e, and pushes it away from C_{avg}.

The distance to this embedding vector can then be used to extend the Euclidean Distance loss function with a constant scaling factor r, as seen in Equation (7).

$$loss_{pnp}(e, c_e) = loss_{euclid}(e, c_e) - r \cdot |e - avg(C)|^2 \quad (7)$$

Figure 9 illustrates the behaviour of $loss_{pnp}$. Note that different from the previously shown figures, Figure 9 shows the loss as a function of the output embedding e. Similar to Euclidean Distance loss, the loss decreases as e moves closer to its class embedding c_e. At the same time, the loss increases if e moves too close to the average class embedding $avg(C)$.

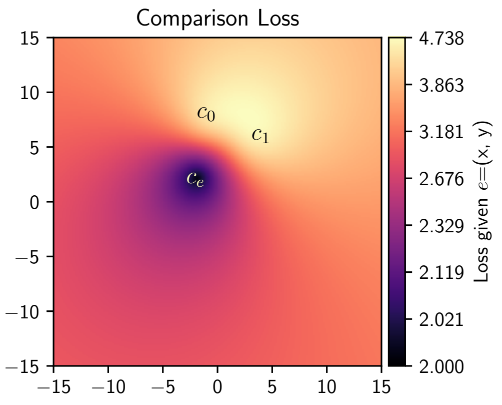


Fig. 10: $loss_{comparison}$ as a function of e given C, similarly to Figure 9. Note that the loss as a function of c_e would be largely indistinguishable from $loss_{euclid}$.

5) Comparison Loss: Another way to keep output embeddings away from the wrong class embeddings is to have a loss function that compares the distance from the output embedding to each of the other class embeddings.

$$loss_{comparison}(e, c_e) = \sum_{k \in C \setminus \{c_e\}} \frac{|e - c_e|^2}{|e - c_e|^2 + |e - k|^2} \quad (8)$$

In Equation (8), two kinds of squared distances are used from the output embedding: the distance to the correct class embedding $|e-c_e|^2$ and the distance to another class embedding $|e-k|^2$. What is then penalised by the loss function is when the distance to the correct class embedding is a more significant part of their sum. Low values of the loss functions are thus found when the fraction is dominated by the large distances to the incorrect class embeddings.

Comparison loss is not used in the experiment section as preliminary testing found it too time-consuming without yielding an apparent improvement. This leads to a key consideration in the design of the loss functions: the complexity of the computation. This complexity mainly involves the number of output embeddings $|E|$ and the number of classes $|C|$. Note that the number of output embeddings is equivalent to the number of images used during training as each image results in a single output embedding. As the loss functions are defined for each output embedding, the complexities include a factor of $|E|$ from the implicit summation of the loss.

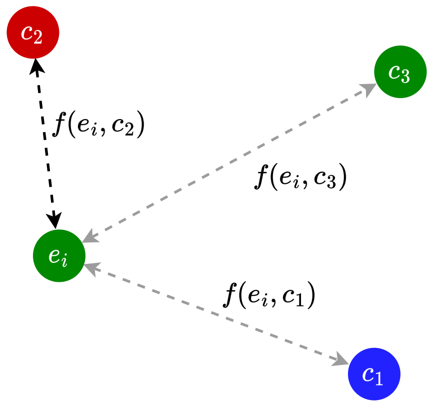


Fig. 11: Illustration of the nearest neighbour classifier used with class embeddings C. Note here that while the true label of the output embedding e_i is c_3 (green), the nearest neighbour classifies e_i as belonging to c_2 (red) as it is closer using the distance function f (for illustration purposes, f is Euclidean distance).

From the definition of Comparison loss and the implicit summation, a time complexity $O(|E| \cdot |C|)$ is found for each epoch of training. Because this is too time-consuming, the other loss functions are designed to have a time complexity of $O(|E|)$. Class Proximity loss calculates all the distances between class embeddings (see Equation (4)) and has a time complexity of $O(|E|+|C|^2)$ when implemented with the second term of its definition computed separately from the first, as it is independent of $|E|$. $|E|$ is however the dominant part of the time complexity as the datasets used in Section IV-A have 60,000 images or more and no more than 100 classes.

D. Classifiers

For this project, nearest neighbour classifiers using various distance metrics are used. The nearest neighbour classifier is defined by Equation (9). The nearest neighbour classifier, when given a point e and a reference set S, finds the point $s \in S$ which minimises the distance metric f.

$$NN_{f,S}(e) = \arg\min_{s \in S} f(e, s) \quad (9)$$

The distance function f may be selected independent of the loss function for which the network is trained, albeit it may reduce the accuracy of the model if selected carelessly.

Figure 11 shows the intuition of the nearest neighbour classifier, while also highlighting a possible flaw of the classifier. When presented with a point, e_i with a true label c_3 (green), the nearest neighbour classifier classifies this point as belonging to c_2 (red), since this class embedding is the closest in terms of the metric function f. This results in a misclassification of the output embedding e_i. The nearest neighbour classifier thus relies directly on the distances in the embedding space with regards to a distance metric.

$$f_{cos}(e, s) = -cos(e, s) \quad (10)$$

$$f_{euclidean}(e, s) = |e - s| \quad (11)$$

The two distance functions used by the Moving Targets Model are given in Equation (10) and Equation (11). In addition, both the class embeddings C and class centres G are used as reference sets. This results in four distinct nearest neighbour classifiers.

Class centres are mainly included as a reference set to allow for comparison with a traditional neural network. Class centres also enable the distinction between the class embedding being badly positioned and output embeddings not being properly separated.

E. Training the model

This section roughly outlines the training loop (generalised for batches) of the Moving Targets Model. The algorithm can be seen in Algorithm 1.

The model m is first initialised with random values for class embeddings C and weights for the neural network m_w. A batch B from training-set D_t is sampled along with the corresponding set of targets T. Then, a feed forward computation of B is performed using network model m. Model m returns the output embeddings E and class embeddings C. Using loss function L, a loss is calculated for the whole batch, and the weights of the network and the class embeddings are then updated to minimise the loss. For evaluation of the current epoch, a feed forward computation of the validation-set D_v is performed. The output of this feed forward computation is then passed to the classifier CL and the accuracy is calculated and subsequently returned.

Algorithm 1: Training loop generalised to batches.

```
Input: Network $m$, Loss function $L$, Classifier $CL$
// Train
for $(B, T) \in D_t$ do
    $E, C \leftarrow$ feed forward $m(B)$;
    $loss \leftarrow L(E, C, T)$;
    update $m_w$ and $C$ to minimise $loss$;
end

// Evaluate
$E, C \leftarrow$ feed forward $m(D_v)$;
$accuracy \leftarrow CL(E, C)$;
return $accuracy$;
```

IV. Experiments

To empirically evaluate the Moving Targets Model, experiments are performed on popular image classification datasets. Image transformations and tuning are used for the trained models. Several variations using the loss functions described in Section III-C are used to compare the effects of changing the loss function. As a baseline, models not using moving targets are also trained. State-of-the-art methods are used for comparison with the results of the Moving Targets Model.

A. Datasets

To measure the stability of the models as the complexities of the datasets increase, experiments are performed using several datasets. The complexity of each dataset is related to: the number of classes present, going from one-channel images (greyscale) to three-channel images (RGB), and increased visual complexity of the images such as more involved backgrounds and other objects being present. This section presents the datasets used in the experiments. An example image from each dataset can be seen in Figure 12.

1) MNIST: The MNIST dataset consists of images of handwritten digits from 0-9 at an image resolution of 28x28 pixels [34]. The training-set contains 60,000 images and the test-set contains 10,000 images. All images are in greyscale. The highest reported accuracy in standard image classification on the test-set is 99.87%[7][13]. Due to the simplicity of the MNIST dataset, high accuracies are achievable for even simple, convolutional neural network models.

2) Fashion-MNIST: Fashion-MNIST (hereafter F-MNIST) is a dataset consisting of greyscale images of clothing from the webshop Zalando [50]. Similar to MNIST, the training-set contains 60,000 images and the test-set contains 10,000 images. While still relatively simple, F-MNIST provides a more difficult benchmark for classification models than MNIST. The highest reported accuracy for an image classification model on F-MNIST is 99.06% [6][45].

3) Kuzushiji-MNIST: Kuzushiji-MNIST (hereafter K-MNIST) is a dataset similar to MNIST and F-MNIST but with 10 classes of Japanese cursive characters [15]. Similar to MNIST and F-MNIST, the training-set contains 60,000 images and the test-set contains 10,000 images. In terms of difficulty, it is easier than F-MNIST, but slightly more difficult than MNIST. The highest accuracy reported on this dataset is 99.34% [2].

4) CIFAR10: The CIFAR10 dataset consists of RGB images from 10 different classes of vehicles and animals [32]. The image resolution is 32x32 pixels. The training-set contains 50,000 images and the test-set contains 10,000 images. The highest reported accuracy on the test-set in standard image classification is 99.5%[3][16].

5) CIFAR100: The CIFAR100 dataset is similar to CIFAR10 but with 100 classes [32]. The training-set contains 50,000 images and the test-set contains 10,00 images, so this means that the image-per-class for CIFAR100 is a tenth of that of CIFAR10. This results in a significantly harder dataset with ten times the classes and only one-tenth of the images for each class when compared to CIFAR10. The highest accuracy reported for this dataset is 96.08% [5][19].

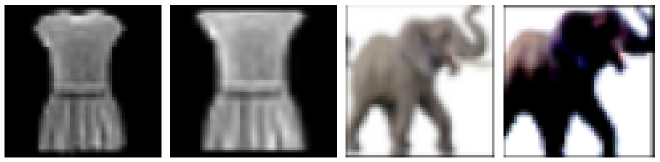
B. Transformations

To improve the performance of neural networks, the input images can be transformed or augmented. For the experiments presented here, only image transformation is used, i.e. no augmentation. The transformations used vary slightly between the datasets. The transformations are composed of resizing, cropping, and normalisation. The resize transformation resizes

(a) ”9” **(b)** ”pullover” **(c)** ”wo”

(d) ”frog” **(e)** ”tiger”

Fig. 12: Example of an image from MNIST, Fashion-MNIST, Kuzushiji-MNIST, CIFAR10, and CIFAR100, respectively.



(a) ”Dress”, normal **(b)** ”Dress”, transformed **(c)** ”elephant”, normal **(d)** ”elephant”, transformed

Fig. 13: Example image transformations from F-MNIST and CIFAR100, respectively. The transformations shown correspond to the transformations described in Section IV-B. Note: Image dimensions are not accurate.

the images to a resolution of 44×44, and the crop transformation performs a centre crop to a resolution of 36×36. The normalisation transformation uses two values to perform a shift-scale transformation [8]. While some of the best image classification models such as ImageNet and ResNet use large resize and cropping resolutions in their experiments [25], the experiments described in this paper use smaller resize and crop resolutions. This is due to the memory requirement and computational time needed for higher resolutions.

Since the MNIST-based data sets are greyscale images and the CIFAR-based data sets are coloured images, the normalisation transformations between these differ. The normalisation is done on each layer in the images, so the greyscale images need a single mean and standard deviation, whereas the coloured images need three means and three standard deviations.

Each image in the datasets is first resized, then cropped, and finally normalised. The transformed image is then used for training, thus the transformations standardise the images rather than provide more training data. The exact transformations can be seen in Table 2. An example of applying the transformations for F-MNIST and CIFAR100 can be seen in Figure 13.

C. Models

To examine the effect of using the Moving Targets Model, both models which use it and models that do not are trained. This is done as fairly as possible so as to not skew the results towards one model or the other. Since the Moving Targets Model defines a neural network and classifier, several combinations must be tested to properly observe the effects of the model.

Dataset	Mean	Standard Deviation
MNIST	33.3149	78.5639
Fashion-MNIST	33.3149	78.5639
KMNIST	33.3149	78.5639
CIFAR10	(0.491, 0.482, 0.447)	(0.202, 0.200, 0.201)
CIFAR100	(0.507, 0.487, 0.441)	(0.201, 0.198, 0.202)

Table 2: The mean value and standard deviation of each colour channel that is used for the normalisation transformation for the five datasets. The resize and crop resolutions for all datasets are 44×44 and 36×36 respectively. Notice, that the MNIST-based datasets use the exact same transformations. For CIFAR10 and CIFAR100, the means and standard deviations vary slightly.

	ResNet18	ResNet50	ResNet101
Euclidean	RN18-E	RN50-E	RN101-E
Cosine	RN18-C	RN50-C	RN101-C
Class proximity	RN18-CP	RN50-CP	RN101-CP
Push and pull	RN18-PP	RN50-PP	RN101-PP
Pure	RN18-P	RN50-P	RN101-P

Table 3: The combinations of models used for training and comparing the effects of the Moving Targets Model. The RN# part of the name refers to the size of the pre-trained network, while the one or two letter at the end represent the loss function. Pure uses a cross-entropy loss function.

For the neural networks, pre-trained neural network models tailored for image classification are used. During the experiments, variations of the ResNet image classification architecture [25], which has seen prolific use within the image classification community, are used. For the purposes of testing the Moving Targets Model, three variations of the ResNet architecture are selected based on the size and complexity: ResNet18, ResNet50, and ResNet101.

Unaltered versions of the three pretrained networks are trained along with the altered versions which use the Moving Targets Model architecture. The trained ResNet models where the Moving Targets Model architecture is not used are referred to as ”pure”. The pure ResNet architecture uses a linear layer and argmax as its classifier.

	RN#-P	RN#-E	RN#-C	RN#-CP	RN#-PP
Epoch	✓	✓	✓	✓	✓
Batch	✓	✓	✓	✓	✓
Learning rate (lr)	✓	✓	✓	✓	✓
Dimension (d)		✓	✓	✓	✓
Scalar (r)				✓	✓

Table 4: The hyperparameters for the models shown in Table 3. Note that the RN# represents the ResNet model used, but since there is no difference in hyperparameters between different ResNet models, they are generalised.

For the models using moving targets, the classifiers are the nearest neighbour classifier variations discussed in Section III-D. This means that for the models using moving targets, there are 12 configurations to test; namely, the combinations from the three neural network choices (ResNet18, ResNet50, ResNet101), and each of the four loss functions (Euclidean Distance, Cosine Distance, Class Proximity, and Push and Pull). For the pure models, there are the three aforementioned configurations, each using a different ResNet model. In total,

	MNIST	F-MNIST	K-MNIST	CIFAR10	CIFAR100
CTM [24]	99.33	91.18	96.03	-	-
ResNet18 [21]	99.64	93.97	98.63	92.93	72.54
ResNet50 [21]	99.56	93.81	98.07	93.78	74.16
ResNet101 [21]	99.54	93.74	98.15	93.40	74.74
pFedBreD-ns-mg [42]	92.47	**99.06**	-	80.63	-
VGG-5 (Spinal FC) (Best) [28]	99.72	94.68	99.15	91.40	64.77
VGG8B (Cross-Entropy) [38]	99.60	94.34	97.78	91.60	70.70
(3x1024 / 3x3000) MLP [38]	99.32	91.40	92.74	67.70	41.10
μNet cont. 1st iter [22]	99.75	-	98.68	-	94.60
HVC Ensemble (Max) [14]	**99.87**	93.89	-	89.23	64.15
Fine-tuned DARTS (+ cutout) [46]	-	96.91	-	97.52	84.10
ViT-H/14 [17]	-	-	-	99.50	94.55
EffNet-L2 + SAM [20]	-	-	-	**99.70**	**96.08**
shake-shake-26 2x96d (S-S-I) (+ cutout) [1]	99.76	96.40	**99.34**	97.10	-
RN18-PP	99.56	91.20	96.68	70.23	32.94
RN50-P	99.37	91.36	96.58	69.99	36.97
RN18-C	99.55	90.75	97.05	70.61	36.71
RN18-P	99.50	90.74	96.80	72.60	39.19

Table 5: Test accuracy of a selection of the state-of-the-art methods on five datasets (first 14 models, above the horizontal line) [37]. The results of the state-of-the-art models are based upon the reported results in their respective publications. The models with the highest accuracy for each dataset are bold. The best performing models using the Moving Target Model are underlined. Four models are selected from the results of this paper. These are seen below the line and are chosen by the best performance for each datasets, always using the classifier which achieves the highest accuracy.

15 different model configurations are tested for each dataset. The different model configurations can be seen in Table 3.

Since all models have to be trained, finding appropriate hyperparameters is necessary. The hyperparameters for the models can be seen in Table 4, where the models using Class Proximity loss or Push and Pull loss have an additional hyperparameter specifically for adjusting the loss function. The pure models do not have a dimension hyperparameter, since the dimensionality of the latent space is decided by the pre-trained ResNet structure. For the models using moving targets, there is a dimensionality hyperparameter deciding the dimensionality of the embedding space. Also note that all models require a learning rate, a batch size, and a number for maximum epochs to be trained.

D. Experiment setup

The search for optimal hyperparameters is done using RayTune, a framework for experiments and hyperparameter tuning [36]. Experiments are conducted using RayTune with HyperOpt as the optimiser. HyperOpt provides derivative-free optimisation given search spaces of hyperparameters [10]. RayTune also provides different trial schedulers for terminating low performing trials, and early stopping mechanism. For trial scheduling, the Async Hyper Band Scheduler (ASHA) is used [35]. For early stopping, a simple plateau stopper is used to terminate trials when the accuracy plateaus. Hyperparameter search is done over 50 trials per experiment.

In total, 75 experiments are run. The following sections consider only the best models found during each experiment (see Appendix A).

E. Performance results

Since the purpose of this project is to investigate the details of the Moving Targets Model, several metrics are recorded during training, such as changes to accuracy, class embeddings, and output embeddings over time. While accuracy is less important regarding the inquiry of this project itself, it is important that the accuracy is at least better than random guessing, as there otherwise may be errors in either the model or the setup. If the accuracies of the trained models are disproportionately low compared to other methods on the datasets tested for this paper, the confidence regarding the remaining analysis of the poorly performing models is harmed.

Table 5 shows a selection of state-of-the-art models [37]. The bottom four models (RN18-PP, RN50-P, RN18-C, and RN18-P) represent the best of the models trained in this paper as selected from Table 3. While the accuracy of the Moving Target Model is not expected to compete with these state-of-the-art models, it is interesting to see how close or far the model compares to the currently best performing models on each dataset.

The Moving Target Model appears to perform within the state-of-the-art models on the MNIST-based datasets, although it does not beat any of them. For the CIFAR-based datasets, the Moving Targets Model performs less close to the state-of-the-art models, save the (3x1024 / 3x3000) MLP [38], for which the Moving Targets Model outperforms on CIFAR10, and is close to the outperforming on CIFAR100. The reason for the relatively worse results on the CIFAR-based datasets may be because of the increased difficulty of these datasets and limitations to the tuning setup.

Regarding the effect of varying the complexity of the pre-trained models, from Figure 14 it can be seen that for the F-MNIST dataset, while not always the case, the models using the smaller ResNet18 neural network has a tendency to perform better than those using either the ResNet50 or ResNet101. As for the other datasets the models are tested on, ResNet18 more clearly outperforms as shown in Tables 6 and 7.

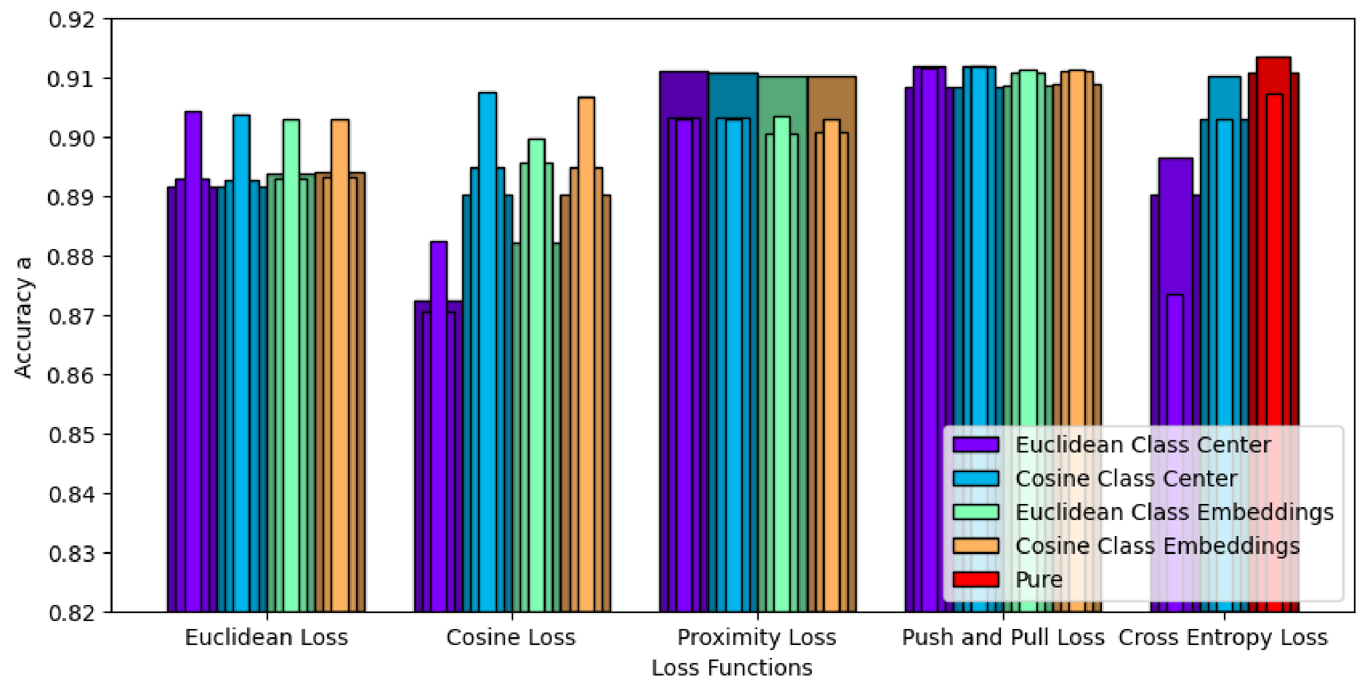


Fig. 14: Bar chart showing the accuracy for each model at the best epoch for the F-MNIST dataset using the classifiers described in Section III-D. The bars with the smallest widths represent the models using the small ResNet pre-trained model, ResNet18. The second smallest width represents models using ResNet50, while the largest width represents models using ResNet101. By virtue of the setup, the Cross Entropy loss function used by the pure set-up is not compatible with classifiers using class embeddings since these are only defined for models using moving targets.

The exceptions to the smaller ResNet models performing better are for the pure set-up, where ResNet50 performs significantly better, and for the Class Proximity loss function where ResNet101 performs, by far, the best. While the pattern appears consistent, an explanation for the difference in the optimal size of the pre-trained model when regarding the pure set-up versus using the Moving Target Model is not clear.

In Table 6 and Table 7, all the accuracies for each model can be seen.

F. Attributes of the loss functions

The following details notable observations found for the loss functions used during the experiments. The plots depicted here only use data from tests on the F-MNIST dataset unless otherwise stated. All plots for all tested datasets may be found in Appendix B. Since the loss functions are designed with different goals in mind, the same analysis does not fit all, and thus the observations detailed depend on the purposes of the loss function for which they are observed. As an example, consider the purpose of Euclidean Distance loss which is: to move points of the same class closer to each other. In relation to this, it is relevant to consider whether the cluster sizes of the actual clusters are reduced during training. However, Cosine Distance loss does not inherit this principle of reducing cluster sizes and this must be taken into consideration as well.

1) Euclidean Distance loss: The reasoning for using Euclidean Distance as a loss function is to compress the clusters under the assumption that the reduced cluster size σ reduces the confusion between classes. Given that the purpose of the Euclidean Distance loss function is to compress the clusters for each class, it is relevant to inspect whether this property can be observed during training. Comparing cluster size σ with the accuracy during training can provide insight into whether the compression of clusters is a beneficial property for achieving increased accuracy. While not part of the loss function directly, it is also relevant to consider the distance between the classes and the global centre, referred to as the centre distance ρ, as it may give some idea as to how much the clusters overlap.

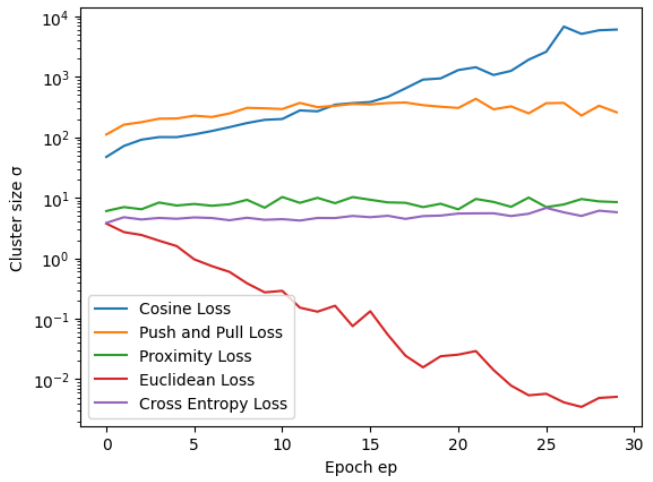


Fig. 15: The cluster size σ for the F-MNIST dataset using the loss functions as described in Section IV-C. The y-axis uses a logarithmic scale (ln). The x-axis shows the epoch during training for which the value is collected. All results are using the test-set of the dataset.

MNIST	cos	cos$_{ce}$	euc	euc$_{ce}$	pure
RN18-E	99.46%	99.47%	99.48%	99.47%	-
RN50-E	99.50%	99.50%	99.50%	99.50%	-
RN101-E	99.38%	99.38%	99.38%	99.38%	-
RN18-C	99.54%	99.55%	98.62%	99.34%	-
RN50-C	99.43%	99.43%	96.73%	99.20%	-
RN101-C	99.36%	99.37%	97.30%	99.09%	-
RN18-CP	99.41%	99.41%	99.42%	99.41%	-
RN50-CP	99.17%	99.17%	99.16%	99.14%	-
RN101-CP	99.37%	99.37%	99.37%	99.37%	-
RN18-PP	99.55%	99.55%	**99.56%**	99.55%	-
RN50-PP	99.45%	99.45%	99.45%	99.45%	-
RN101-PP	99.45%	99.45%	99.45%	99.45%	-
RN18-P	99.47%	-	99.08%	-	99.50%
RN50-P	99.35%	-	99.04%	-	99.37%
RN101-P	99.23%	-	98.92%	-	99.25%

F-MNIST	cos	cos$_{ce}$	euc	euc$_{ce}$	pure
RN18-E	90.39%	90.30%	90.44%	90.31%	-
RN50-E	89.26%	89.33%	89.29%	89.31%	-
RN101-E	89.17%	89.40%	89.15%	89.39%	-
RN18-C	90.75%	90.69%	88.23%	89.97%	-
RN50-C	89.48%	89.49%	87.04%	89.58%	-
RN101-C	89.03%	89.03%	87.25%	88.22%	-
RN18-CP	90.30%	90.29%	90.31%	90.34%	-
RN50-CP	90.32%	90.08%	90.33%	90.06%	-
RN101-CP	91.09%	91.02%	91.10%	91.02%	-
RN18-PP	91.20%	91.15%	91.17%	91.15%	-
RN50-PP	91.18%	91.11%	91.18%	91.08%	-
RN101-PP	90.84%	90.89%	90.85%	90.87%	-
RN18-P	90.30%	-	87.36%	-	90.74%
RN50-P	91.03%	-	89.64%	-	**91.36%**
RN101-P	90.30%	-	89.02%	-	91.07%

K-MNIST	cos	cos$_{ce}$	euc	euc$_{ce}$	pure
RN18-E	96.89%	96.86%	96.81%	96.81%	-
RN50-E	94.65%	94.67%	94.64%	94.70%	-
RN101-E	90.37%	90.18%	90.41%	90.17%	-
RN18-C	**97.05%**	**97.05%**	92.56%	96.17%	-
RN50-C	96.43%	96.43%	88.06%	95.88%	-
RN101-C	96.42%	96.41%	85.96%	95.71%	-
RN18-CP	96.89%	96.89%	96.84%	96.84%	-
RN50-CP	93.50%	93.51%	93.38%	93.37%	-
RN101-CP	96.47%	96.47%	96.48%	96.46%	-
RN18-PP	96.98%	96.98%	96.98%	96.98%	-
RN50-PP	96.69%	96.69%	96.69%	96.69%	-
RN101-PP	96.46%	96.45%	96.45%	96.45%	-
RN18-P	96.58%	-	92.73%	-	96.80%
RN50-P	96.10%	-	93.43%	-	96.58%
RN101-P	96.25%	-	94.78%	-	96.40%

Table 6: Accuracy of each model, for the epoch where it scored the highest accuracy on MNIST-based test-data.

CIFAR10	cos	cos$_{ce}$	euc	euc$_{ce}$	pure
RN18-E	71.53%	71.46%	71.48%	71.41%	-
RN50-E	67.41%	67.42%	67.37%	67.32%	-
RN101-E	65.18%	64.90%	65.13%	64.93%	-
RN18-C	70.52%	70.61%	65.18%	69.83%	-
RN50-C	66.65%	66.65%	59.77%	62.85%	-
RN101-C	63.94%	63.89%	56.44%	61.61%	-
RN18-CP	70.54%	70.60%	70.53%	70.52%	-
RN50-CP	69.03%	69.04%	68.72%	68.75%	-
RN101-CP	69.33%	69.30%	69.29%	69.28%	-
RN18-PP	70.22%	70.12%	70.15%	70.23%	-
RN50-PP	67.98%	67.96%	67.81%	67.79%	-
RN101-PP	68.38%	68.34%	68.37%	68.26%	-
RN18-P	**72.60%**	-	69.93%	-	72.47%
RN50-P	69.18%	-	67.70%	-	69.99%
RN101-P	67.17%	-	65.65%	-	69.01%

CIFAR100	cos	cos$_{ce}$	euc	euc$_{ce}$	pure
RN18-E	36.33%	36.71%	33.60%	35.61%	-
RN50-E	33.37%	34.62%	32.11%	32.65%	-
RN101-E	31.92%	32.53%	30.75%	31.23%	-
RN18-C	35.89%	36.71%	31.97%	35.71%	-
RN50-C	33.13%	32.83%	20.52%	29.78%	-
RN101-C	31.60%	31.73%	22.04%	30.13%	-
RN18-CP	36.10%	36.22%	34.34%	34.69%	-
RN50-CP	34.61%	34.66%	33.24%	33.33%	-
RN101-CP	28.23%	28.13%	25.44%	28.19%	-
RN18-PP	32.80%	32.88%	31.91%	32.94%	-
RN50-PP	31.75%	32.07%	30.67%	31.21%	-
RN101-PP	28.84%	29.19%	27.79%	28.44%	-
RN18-P	38.67%	-	37.85%	-	**39.19%**
RN50-P	35.25%	-	35.15%	-	36.97%
RN101-P	34.03%	-	33.84%	-	35.40%

Table 7: Accuracy of each model, for the epoch where it scored the highest accuracy on CIFAR-based test-data.

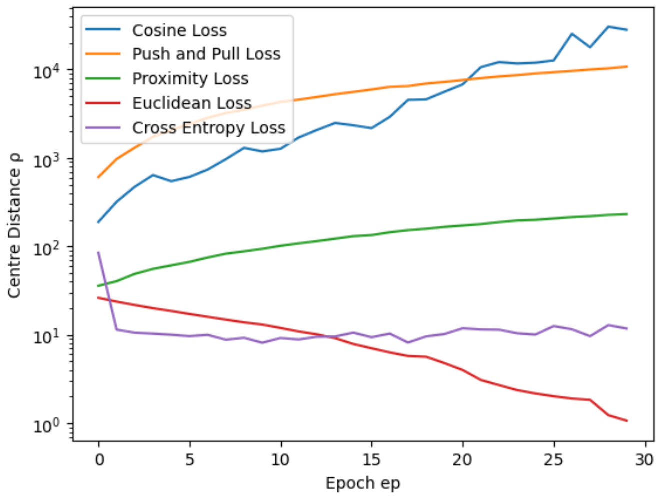


Fig. 16: The mean centre distance ρ for the F-MNIST dataset using the loss functions as described in Section IV-C. The y-axis uses a logarithmic scale (ln). The x-axis shows the epoch during training for which the value is collected. All results are using the test-set of the dataset.

On Figure 15, the cluster size, aggregated over all classes, can be seen for F-MNIST. For the Euclidean Distance loss function, the cluster size, and therefore the median distance between output embeddings and class centres, decreases significantly (exponentially) during training.

Comparing cluster sizes in Figure 15 to the mean centre distances in Figure 16, shows that the class centres are unlikely to overlap. This can be seen in Figure 17. While the centre distance does decrease during training, this distance is initially greater than the cluster size from Figure 15 and it decreases slower. As such, it appears that the class centres, and, by extension, the clusters are far enough away from the global centre that they are not overlapping over the global centre. While this does indicate that the classes are not overlapping to a large degree, as, if they did, the global centre would be placed within the cluster, there may still be significant overlap between some of the clusters.

2) Cosine Distance loss: While the other loss functions described in Section III-C work on the distances between output embeddings, Cosine Distance loss instead works on the direction of the output embeddings as interpreted using a reference point. The intuition is that if points belonging to the same class move in the same direction, they eventually become linearly separable, as they are moved more-so away

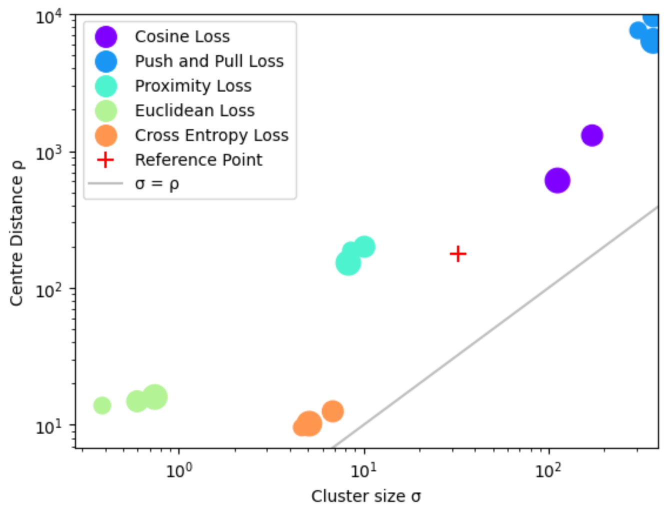


Fig. 17: The centre distance ρ compared to the cluster size σ. A line is added to shown when the two distances are equal. Anything above the line has a centre distance greater than cluster size. The different point sizes indicate the size of the pre-trained network, such that the smallest point is ResNet18, the second smallest point is ResNet50, and the largest point is ResNet101. The reference point is the global centre for the first epoch.

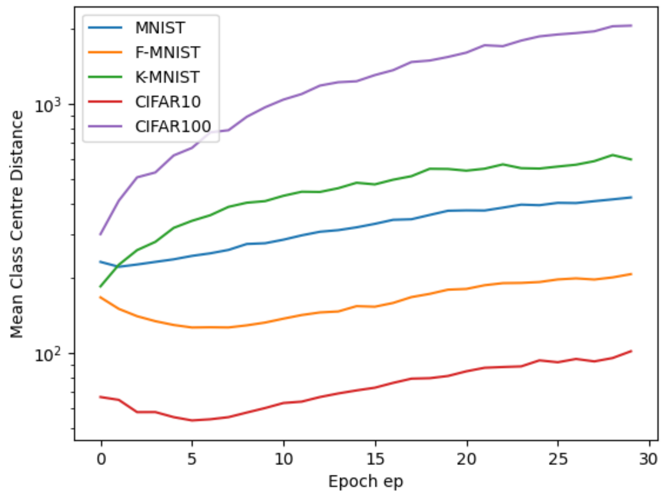


Fig. 18: The mean distance between all class centres as a function of epochs during training. The data is for the validation set used during training, with class centres calculated as described in Section III-B. Here, only the ResNet18 model is selected for each dataset and only using Class Proximity loss as described in Section III-C.

from incorrect classes. Thus, the purpose of Cosine Distance loss is to move the output embeddings so far away from the reference point that the in-cluster distances become negligible compared to the distances between clusters.

From Figure 16 and Figure 15 one can observe that the embeddings move both away from the global centre but also away from the class centre when trained. While the increase in cluster size may appear to clash directly with the purpose of the Cosine Distance loss function, the increase in median distance is less than the increase in centre distance. Also, since the Cosine Distance loss is normalised for the direction vectors, it does not regard the difference in position between output embeddings, and as such some output embeddings may be placed much further away from their respective class centre but along the same direction.

While the Cosine Distance loss function is the one most similar to the traditional neural network (see Section V-B), it is also the loss function consistently yielding the lowest accuracies across datasets as seen in Table 6 and Table 7. Especially the combination of a cosine loss during training with a Euclidean Distance nearest neighbour classifier yields the lowest accuracies of all models. This observation follows intuitively from the purposes of the Cosine Distance loss function compared to the purpose of a Euclidean Distance loss function. While the Euclidean Distance loss function, and here also classifier, seeks to group output embeddings closely which belong to the same class, it does not consider the directions of these points. While, for great distances, the direction converges when using a Euclidean-based loss function, for small distances, the difference in direction between points belonging to the same class may be great and thus not a suitable classification attribute. Oppositely, the purpose of the Cosine Distance loss function considers only direction and not distance, and so, not surprisingly, the combination of those two yield an incompatible result. For the opposite case, however, where the Euclidean Distance loss function is used and the Cosine Distance classifier, the resulting accuracies are indistinguishable from Euclidean on Euclidean.

3) Class Proximity loss: The purpose of Class Proximity loss is to address one of the potential issues with the Euclidean Distance loss function, which is that all points across classes may converge on a single point, thus reducing the effectiveness of nearest neighbour classification.

$$loss_{prox}(e, c_e) = loss_{euclid}(e, c_e) + r \cdot prox(c_e) \qquad (12)$$

Recall Equation (5) from Section III-C, reproduced here as Equation (12) for readability. Preliminary testing showed that using values for the hyperparameter r in the interval $[10; 1,000]$ meant that the value of the term $prox(c_e)$ had little effect on the overall loss, and, by extension, the gradients. In practice, this meant that using values within such interval produced a loss function almost equivalent to the Euclidean Distance loss function. Instead, a larger interval of $[1,000; 1,000,000]$ for proximity multiplier r was tested and showed to have immediate effect on the distances between class embeddings.

To see whether the Class Proximity loss function behaves as expected, the mean distance between class centres over time can be visualised. This data is shown in Figure 18 for all datasets but only using the ResNet18 pre-trained model. The Class Proximity loss function does increase the distance between class centres, as the output embeddings are pushed away from the class centres. Since this is only part of the Class Proximity loss function, the cluster size of the classes must also be considered, as the distance between class centres is not enough to warrant a successful application of the loss function. From Figure 15, the cluster size for Class Proximity loss cannot be seen to either increase or decrease as the model

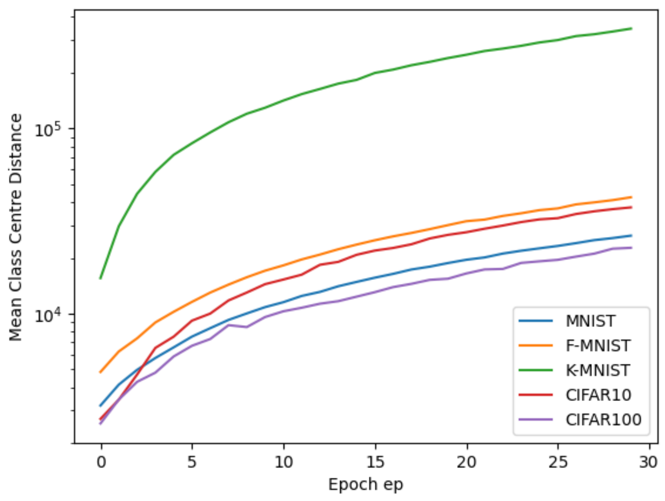


Fig. 19: The mean distance between all class centres as a function of epochs during training. Portraits the same information as Figure 18 but for Push and Pull loss as opposed to Class Proximity loss.

is trained. This means that Class Proximity loss successfully achieves the desired effect of pushing class embeddings away from other class embeddings while keeping the cluster for each class tight.

4) Push and Pull loss: The purpose of the Push and Pull loss function is similar to Class Proximity loss, but with a different approach to keep the cluster for each class tight but at the same time away from other classes. Figure 19 shows, similar to Figure 18, the mean distance between class centres for each epoch. From Figure 19, it is evident that the mean distance between the class centres increases as the model is trained, meaning that the Push and Pull loss function succeeds in separating the classes.

While the pushing mechanism of Push and Pull loss is shown to work, the pulling mechanism must also be confirmed. Confirming this mechanism can be done by observing the change in cluster sizes, which should decrease during training. Figure 15 shows the cluster size during training for the F-MNIST dataset, including the Push and Pull loss function. It can be seen that the Push and Pull loss does not decrease the cluster size during training but instead slightly increases it. This means that the pulling mechanism is not as effective as expected; however, the increase in cluster size is small compared to the mean distance between class centres in Figure 19. This may indicate an imbalance in the tested models regarding the weighting of the pulling and pushing mechanisms of the loss function.

V. Discussion

While developing the Moving Target Model and performing experiments to investigate the workings of the model, several possibly interesting observations have been collected. Here, the more important of such observations are discussed as well as the possible positive or negative effects such observations may have on the results of the model.

A. *Low accuracy of trained models*

An immediate concern regarding the results presented in Section IV-E is the comparatively low accuracy of the models trained compared to the reported results of the state-of-the-art models. While some loss of accuracy can be attributed to a less rigorous and intensive focus on achieving a high accuracy, the concern is particularly relevant for the CIFAR-based datasets. As such, it is relevant to discuss what may have caused the reduction in accuracy. Furthermore, the following sections also discuss the properties of the Moving Targets Model and the use of pre-trained neural network models.

One of the reasons for the low accuracies may be due to the lack of better transformations. The transformations described in section IV-B are a compromise of more expensive transformations that would have significantly increased the time needed for the experiments. Transformations used during training of the ResNet architectures as described by He et al. [25] upscale the images using scale augmentation followed by a 224x224 crop of the upscaled image and horizontal flips. Using transformations similar to those by He et al. [25] may improve the accuracy of the Moving Targets Model at the cost of substantially higher computational requirements and time required for training.

B. *Interpretation as neural network layer*

While the Moving Targets Model was initially perceived as being unique from the traditional neural network, it was during development noted that the model may be perceived as a fully connected linear neural network layer using a different operation than dot-product between the input values and weights to calculate the output values. The class embeddings thus correspond to the weights of a linear layer, while the biases do not have an equivalent.

The model using the cosine loss function is the version that most closely resembles a traditional network using a linear layer. The main difference is that using the class embeddings with that loss function works as a linear layer where both the input of the layer and the weights are normalised (see Equation (2)).

C. *Using pre-trained neural networks*

The models that are trained and tested in Section IV-E all use the ResNet architecture with pre-trained weights. The focus of this article is not to train neural networks to perform well on image classification tasks, but rather see the effects of using moving targets within an embedding space. Therefore building and verifying a neural network architecture would be unnecessary. Instead, since the ResNet models are trained and shown to perform well on popular image classification tasks [25], these are used as pre-trained neural networks. Since the ResNet models are trained on images and classes not present in the datasets, the models must be fine-tuned, which is done similarly to normal training of neural networks.

For the purposes of this project, three ResNet pre-trained neural network models are selected with varying complexity. The complexity of these models stem from the number of trainable layers, and models with 18, 50, and 101 trainable layers are selected for the neural network part of the Moving Targets Model. While more complex models should be able to capture more complex patterns, the smallest ResNet model used here, ResNet18, performs better than using the ResNet50 and ResNet101 models on most tested datasets. A possible explanation for the higher accuracies of the models using ResNet18 as opposed to ResNet50 and ResNet101 is the tuning setup used to fine-tune the models. It has previously been conjectured that deeper models have exponentially lower convergence rates [25], and during the experiments, all models are only trained up to at most 30 epochs. A combination of different hyperparameters, transformations, and more training epochs may improve performance for the larger architectures.

Another consideration towards the usage of pre-trained models is the semantic meaning assigned to the last linear layer of such a model, which here is interpreted as an embedding layer. Since the pre-trained models have already learned a semantic meaning to the input images, the expectation may be that images possessing similar attributes are already grouped within the embedding space of the last layer. However, the semantic meaning learned through the pre-training may not uphold this property of tight clustering of images with similar attributes. Instead there may be several such clusters or there may be no distinguishable clusters at all. Since the moving target model relies on moving images into a singular cluster for each class, there may be more tuning of pre-trained networks if there is already a conflicting semantic meaning to the images.

VI. Conclusion

It is shown that the introduced Moving Targets Model allows the semantic representation of images to be changed in the embedding space. Specifically, this change is governed by the loss functions, and it is shown that different loss functions in the Moving Targets Model position output embeddings and class embeddings differently. Four specific loss functions (Euclidean Distance, Cosine Distance, Class Proximity, and Push & Pull) are tested, revealing that their combinations with the Moving Targets Model each give distinct properties to the points within the embedding space. The loss functions described in this paper focus on clustering and manipulating the relative positioning of clusters. The positioning of images within the embedding space can be controlled via different loss functions, allowing for properties other than clustering.

By providing a latent space for image classification, the Moving Targets Model brings a geometric understanding to the domain. This geometric perspective offers new ways to understand and reason about new loss functions, thereby assisting the future development of loss functions.

The ability to reason about loss functions is valuable, but the practicality thereof relies on maintaining a high accuracy. Comparing to state of the art is unreasonable, as tools such as data augmentation are not explored. Instead, the Moving Targets Model should be compared to the baseline of pure ResNet models, which are the original, pre-trained ResNet models that are trained with the same data and under the same conditions as the Moving Targets Model experiments are. The accuracies achievable with the Moving Targets Model are generally similar to those of the baseline pure ResNet models. Thus, modifying a ResNet model to use the Moving Targets Model can be done at minimal cost in accuracy.

As such, introducing the Moving Targets Model to the task of image classification provides a novel, geometric understanding of images within an embeddings space at a minimal penalty to performance with regards to accuracy of predictions. Introducing this geometric understanding allows for new loss functions to be described and reasoned about, opening up further development in the field of machine learning.

A. Future Work

The following topics are identified but are left unexplored. Some topics revolve around further investigation into the workings of the Moving Targets Model as a means to better understand the implication of using a geometric interpretation of images. Some topics revolve around the further development of loss functions to possibly achieve better performance.

The loss functions that are described and examined in this paper are simple by nature. More complex loss functions may be leveraged in order for the Moving Targets Model to possibly achieve increased accuracy on image classification tasks.

Use cases for the geometric interpretation of the embedding space are a relevant next step in the development of the Moving Targets Model. For instance, whether there is, for the clusters created by most of the tested loss functions, a correlation between the position of output embeddings with respect to its class centre, and the certainty of the prediction is left unexplored.

It may also be worth investigating whether the clustering property achieved by several of the loss functions is transferable to classes not included in the training set. This could make the techniques investigated in this paper applicable to Few-Shot Image Classification [4].

In addition, the controlled creation of embeddings may serve as an intermediary encoding for other tasks that do not directly use classification.

Acknowledgment

The authors would like to thank supervisor Christian Schilling for help and guidance throughout the project and Aalborg University for providing resources for research.

References

[1] Github - hysts/pytorch_image_classification: Pytorch implementation of image classification models for cifar-10/cifar-100/mnist/fashionmnist/kuzushiji-mnist/imagenet — github.com. https://github.com/hysts/pytorch_image_classification, 2023. Accessed: 24-05-2023.

[2] GitHub - rois-codh/kmnist: Repository for Kuzushiji-MNIST, Kuzushiji-49, and Kuzushiji-Kanji — github.com. https://github.com/rois-codh/kmnist, 2023. Accessed: 24-05-2023.

[3] Papers with code - cifar10 benchmark (image classification), 2023. https://paperswithcode.com/sota/image-classification-on-cifar10-1 Accessed: 24-05-2023.

[4] Papers with code - cifar10 benchmark (image classification), 2023. https://paperswithcode.com/task/few-shot-image-classification Accessed: 24-05-2023.

[5] Papers with code - cifar100 benchmark (image classification), 2023. https://paperswithcode.com/sota/image-classification-on-cifar100 Accessed: 24-05-2023.

[6] Papers with code - fashion-mnist benchmark (image classification), 2023. https://paperswithcode.com/sota/image-classification-on-fashion-mnist Accessed: 24-05-2023.

[7] Papers with code - mnist benchmark (image classification), 2023. https://paperswithcode.com/sota/image-classification-on-mnist Accessed: 24-05-2023.

[8] Torch.nn.functional.normalize. https://pytorch.org/docs/stable/generated/torch.nn.functional.normalize.html#torch.nn.functional.normalize Accessed: 24-05-2023, 2023.

[9] Bjorn Barz and Joachim Denzler. Hierarchy-based image embeddings for semantic image retrieval. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, jan 2019.

[10] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–115–I–123. JMLR.org, 2013.

[11] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[12] Andriy Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[13] Adam Byerly, Tatiana Kalganova, and Ian Dear. A branching and merging convolutional network with homogeneous filter capsules. *CoRR*, abs/2001.09136, 2020.

[14] Adam Byerly, Tatiana Kalganova, and Ian Dear. No routing needed between capsules, 2021.

[15] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[18] Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. With a little help from my friends: Nearest-neighbor contrastive learning of visual representations, 2021.

[19] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *CoRR*, abs/2010.01412, 2020.

[20] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization, 2021.

[21] Paul Gavrikov and Janis Keuper. CNN filter DB: An empirical investigation of trained convolutional filters. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2022.

[22] Andrea Gesmundo and Jeff Dean. An evolutionary approach to dynamic introduction of tasks in large-scale multitask learning systems, 2022.

[23] Florin. Gorunescu. *Data Mining Concepts, Models and Techniques*. Intelligent Systems Reference Library, 12. Springer Berlin Heidelberg, Berlin, Heidelberg, 1st ed. 2011. edition, 2011.

[24] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. The convolutional tsetlin machine, 2019.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[26] C. Huang, L.S. Davis, and J.R.G. Townshend. An assessment of support vector machines for land cover classification. *International Journal of Remote Sensing*, 23(4):725–749, 2002.

[27] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[28] H M Dipu Kabir, Moloud Abdar, Seyed Mohammad Jafar Jalali, Abbas Khosravi, Amir F Atiya, Saeid Nahavandi, and Dipti Srinivasan. Spinalnet: Deep neural network with gradual input, 2022.

[29] Ivan Karpukhin, Stanislav Dereka, and Sergey Kolesnikov. Exact: How to train your accuracy, 2022.

[30] Jens Keuchel, Simone Naumann, Matthias Heiler, and Alexander Siegmund. Automatic land cover analysis for tenerife by supervised classification using remotely sensed data. *Remote Sensing of Environment*, 86(4):530–541, 2003.

[31] Kamran Kowsari, Mojtaba Heidarysafa, Donald E. Brown, Kiana Jafari Meimandi, and Laura E. Barnes. RMDL. In *Proceedings of the 2nd International Conference on Information System and Data Mining*. ACM, apr 2018.

[32] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Cifar10 and cifar100 dataset. https://www.cs.toronto.edu/~kriz/cifar.html Accessed: 24-05-2023.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[34] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database. http://yann.lecun.com/exdb/mnist/ Accessed: 24-05-2023.

[35] Liam Li, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. *CoRR*, abs/1810.05934, 2018.

[36] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[37] Meta AI. Our mission. https://paperswithcode.com/about#team, 2022.

[38] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals, 2019.

[39] Mahesh Pal and Paul M Mather. An assessment of the effectiveness of decision tree methods for land cover classification. *Remote Sensing of Environment*, 86(4):554–565, 2003.

[40] David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*, pages 310–316. Cambridge University Press, Cambridge, UK, 2 edition, 2017.

[41] Lior Rokach and Oded Maimon. *Decision Trees*, volume 6, pages 165–192. 01 2005.

[42] Mingjia Shi, Yuhao Zhou, Qing Ye, and Jiancheng Lv. Personalized federated learning with hidden information on personalized prior, 2022.

[43] Xu Sun, Bingzhen Wei, Xuancheng Ren, and Shuming Ma. Label embedding network: Learning label representation for soft training of deep networks, 2017.

[44] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[45] Muhammad Suhaib Tanveer, Muhammad Umar Karim Khan, and Chong-Min Kyung. Fine-tuning DARTS for image classification. *CoRR*, abs/2006.09042, 2020.

[46] Muhammad Suhaib Tanveer, Muhammad Umar Karim Khan, and Chong-Min Kyung. Fine-tuning darts for image classification, 2020.

[47] Mofassir ul Islam Arif, Mohsan Jameel, Josif Grabocka, and Lars Schmidt-Thieme. Phantom embeddings: Using embedding space for model regularization in deep neural networks, 2023.

[48] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[49] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[50] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[51] Zelin Zang, Siyuan Li, Di Wu, Ge Wang, Lei Shang, Baigui Sun, Hao Li, and Stan Z. Li. Dlme: Deep local-flatness manifold embedding, 2022.

APPENDIX A
BEST MODEL HYPERPARAMETERS

CIFAR10	Learning Rate	d	r_{CP}	r_{PP}
RN18-E	5.793582×10^{-4}	558	-	–
RN50-E	4.331950×10^{-4}	877	-	–
RN101-E	5.021565×10^{-4}	1027	-	–
RN18-C	5.656482×10^{-4}	1609	-	–
RN50-C	4.970138×10^{-4}	1809	-	–
RN101-C	2.555302×10^{-4}	763	-	–
RN18-CP	1.206190×10^{-3}	1528	260007	–
RN50-CP	6.080175×10^{-4}	1462	553119	–
RN101-CP	5.633006×10^{-4}	1612	754617	–
RN18-PP	5.205251×10^{-4}	1824	-	0.95
RN50-PP	2.515188×10^{-4}	532	-	0.95
RN101-PP	2.926039×10^{-4}	539	-	0.95
RN18-P	9.678284×10^{-4}	-	-	–
RN50-P	1.115847×10^{-3}	-	-	–
RN101-P	1.423183×10^{-3}	-	-	–

CIFAR100	Learning Rate	d	r_{CP}	r_{PP}
RN18-E	2.416471×10^{-4}	1152	-	–
RN50-E	3.710583×10^{-4}	1277	-	–
RN101-E	2.088279×10^{-4}	1659	-	–
RN18-C	2.640890×10^{-4}	974	-	–
RN50-C	5.619728×10^{-4}	1304	-	–
RN101-C	3.762319×10^{-4}	680	-	–
RN18-CP	6.518272×10^{-4}	545	766693	–
RN50-CP	3.340637×10^{-4}	1039	711827	–
RN101-CP	4.464016×10^{-4}	1579	435022	–
RN18-PP	1.925196×10^{-4}	203	-	0.95
RN50-PP	2.436377×10^{-4}	333	-	0.95
RN101-PP	1.982509×10^{-4}	829	-	0.95
RN18-P	5.857224×10^{-4}	-	-	–
RN50-P	5.793152×10^{-4}	-	-	–
RN101-P	5.403263×10^{-4}	-	-	–

Table 8: Hyperparameters for the best models found during hyperparameter search.

MNIST	Learning Rate	d	r_{CP}	r_{PP}
RN18-E	4.409685×10^{-3}	394	-	–
RN50-E	9.346739×10^{-4}	898	-	–
RN101-E	2.698507×10^{-4}	735	-	–
RN18-C	2.173782×10^{-3}	1010	-	–
RN50-C	2.492074×10^{-3}	1990	-	–
RN101-C	1.809809×10^{-3}	1843	-	–
RN18-CP	8.098920×10^{-3}	1862	496999	–
RN50-CP	4.536270×10^{-4}	491	769771	–
RN101-CP	2.018530×10^{-3}	1576	330875	–
RN18-PP	4.662704×10^{-4}	420	-	0.95
RN50-PP	5.256893×10^{-4}	614	-	0.95
RN101-PP	2.066876×10^{-3}	1244	-	0.95
RN18-P	1.255004×10^{-3}	-	-	–
RN50-P	4.195942×10^{-4}	-	-	–
RN101-P	1.251098×10^{-3}	-	-	–

F-MNIST	Learning Rate	d	r_{CP}	r_{PP}
RN18-E	4.453946×10^{-4}	1351	-	–
RN50-E	6.793674×10^{-4}	1230	-	–
RN101-E	6.811457×10^{-4}	1466	-	–
RN18-C	3.411182×10^{-4}	1901	-	–
RN50-C	2.412923×10^{-4}	1291	-	–
RN101-C	3.378574×10^{-4}	1006	-	–
RN18-CP	8.485818×10^{-4}	1247	400738	–
RN50-CP	3.917616×10^{-4}	1259	355301	–
RN101-CP	4.556357×10^{-4}	1455	374363	–
RN18-PP	4.659248×10^{-4}	1285	-	0.95
RN50-PP	5.305126×10^{-4}	897	-	0.95
RN101-PP	1.281051×10^{-3}	458	-	0.95
RN18-P	4.701197×10^{-3}	-	-	–
RN50-P	5.861841×10^{-4}	-	-	–
RN101-P	3.175975×10^{-3}	-	-	–

K-MNIST	Learning Rate	d	r_{CP}	r_{PP}
RN18-E	2.824394×10^{-3}	1669	-	–
RN50-E	1.441133×10^{-3}	910	-	–
RN101-E	1.360239×10^{-3}	662	-	–
RN18-C	1.259559×10^{-3}	843	-	–
RN50-C	3.039139×10^{-4}	597	-	–
RN101-C	9.399527×10^{-4}	1393	-	–
RN18-CP	2.367562×10^{-3}	1280	908089	–
RN50-CP	1.013034×10^{-3}	913	27585	–
RN101-CP	7.484629×10^{-4}	373	187259	–
RN18-PP	3.460508×10^{-3}	1502	-	0.95
RN50-PP	7.090483×10^{-4}	1298	-	0.95
RN101-PP	3.273086×10^{-4}	455	-	0.95
RN18-P	3.815637×10^{-3}	-	-	–
RN50-P	1.501105×10^{-3}	-	-	–
RN101-P	5.718608×10^{-4}	-	-	–

Table 9: Hyperparameters for the best models found during hyperparameter search.

Appendix B
Plots

A. Accuracy

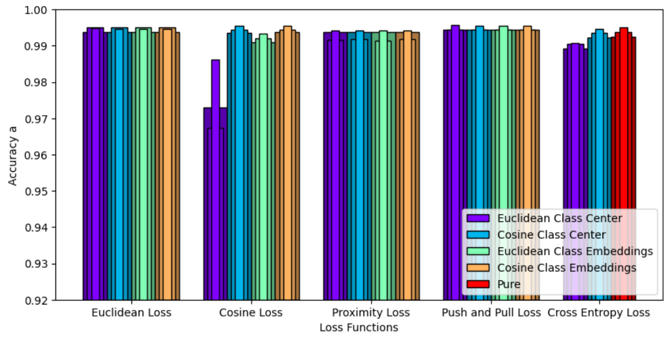


Fig. 20: MNIST

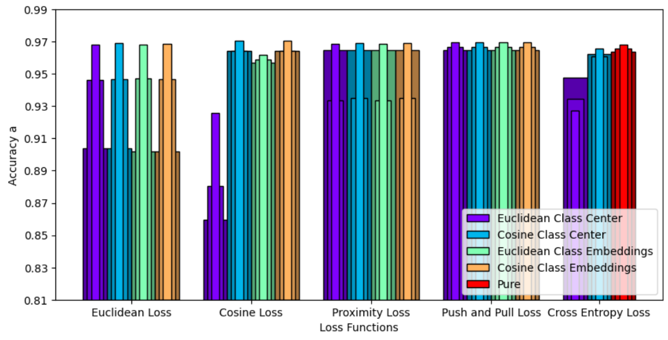


Fig. 21: K-MNIST

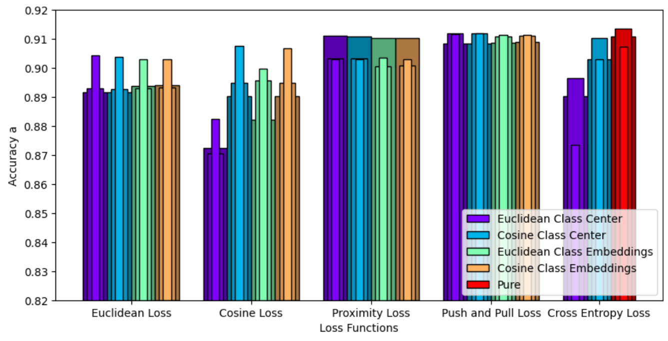


Fig. 22: F-MNIST

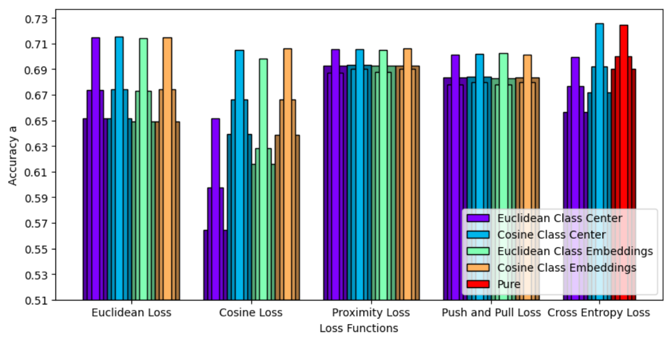


Fig. 23: CIFAR10

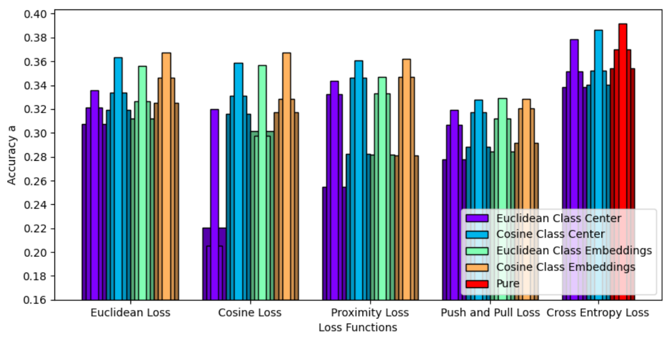


Fig. 24: CIFAR100

B. Accuracy over time

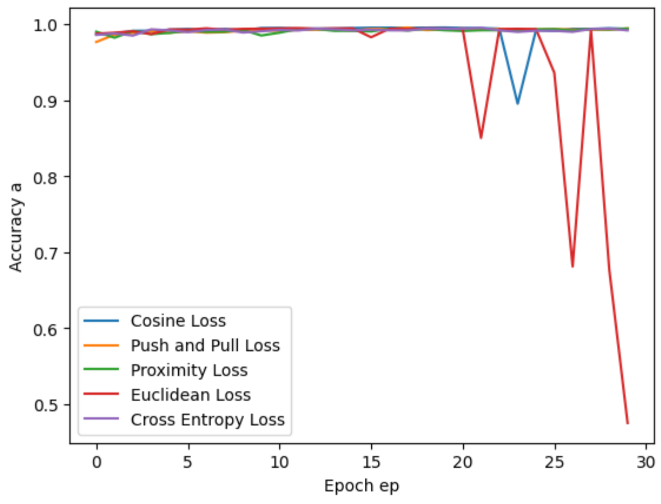


Fig. 25: MNIST

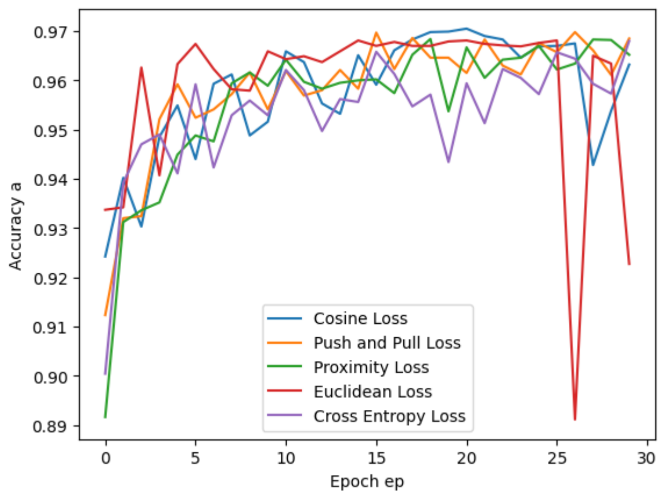


Fig. 26: K-MNIST

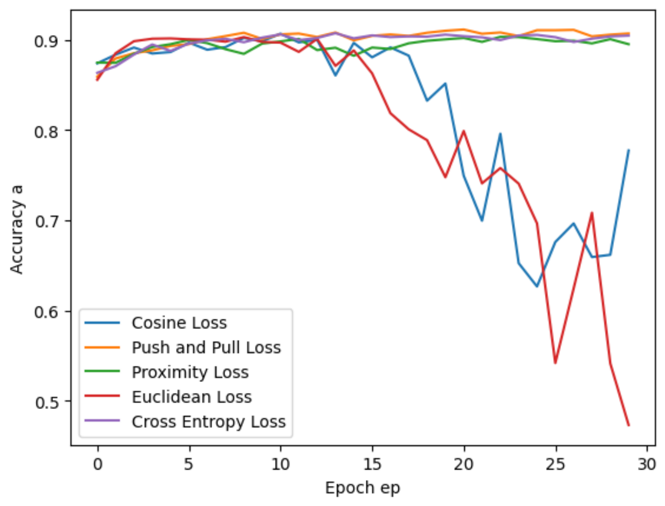


Fig. 27: F-MNIST

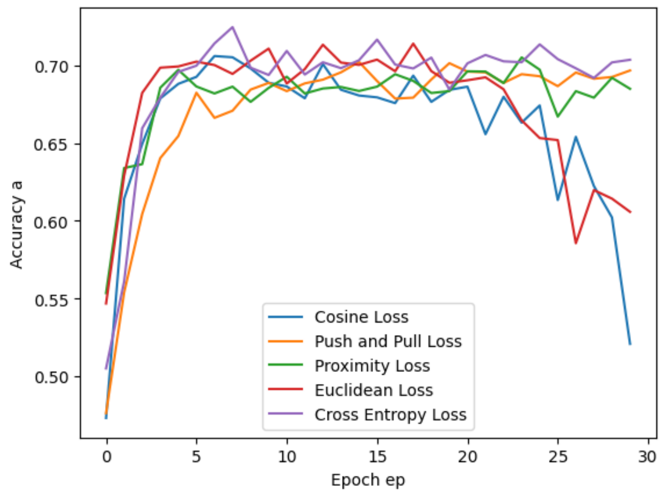


Fig. 28: CIFAR10

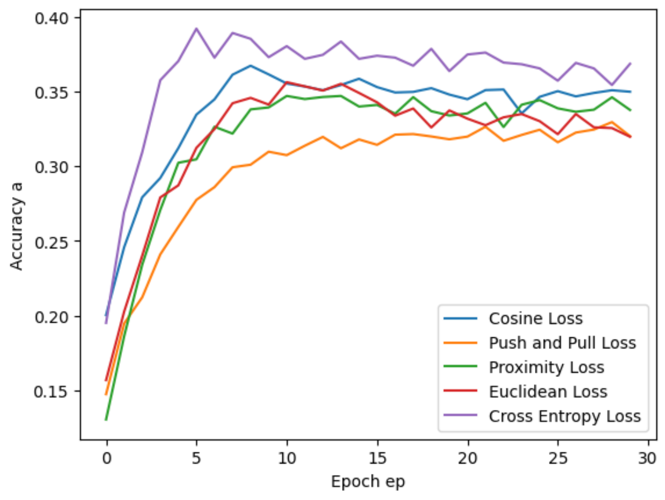


Fig. 29: CIFAR100

C. Centre distance over time

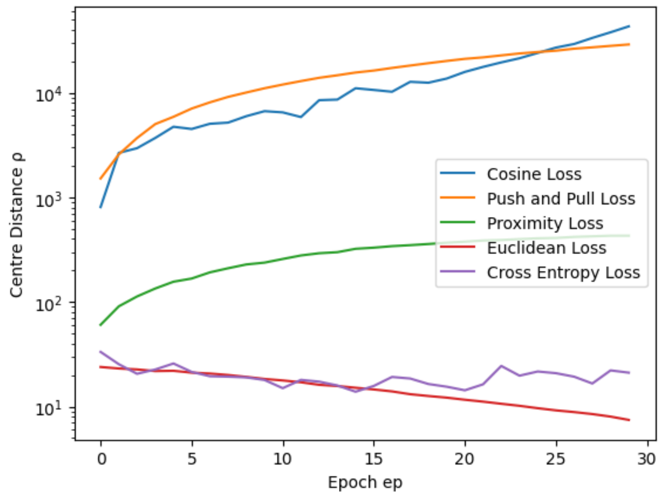


Fig. 30: MNIST

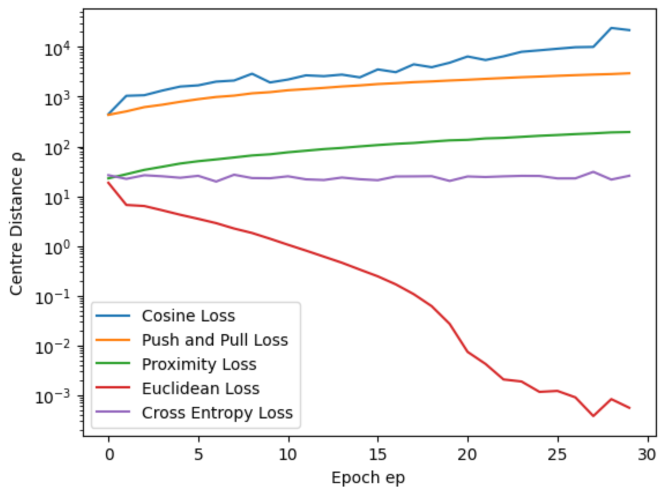


Fig. 31: K-MNIST

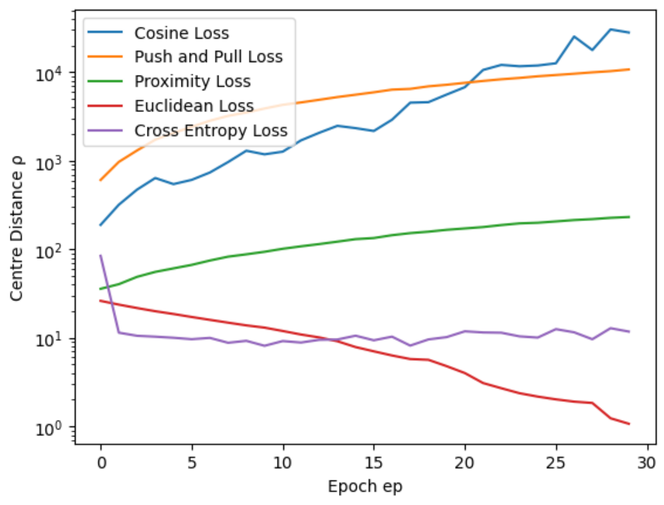


Fig. 32: F-MNIST

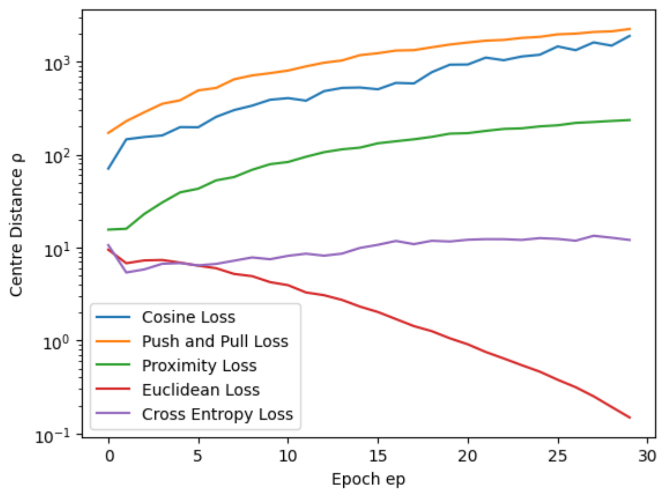


Fig. 33: CIFAR10

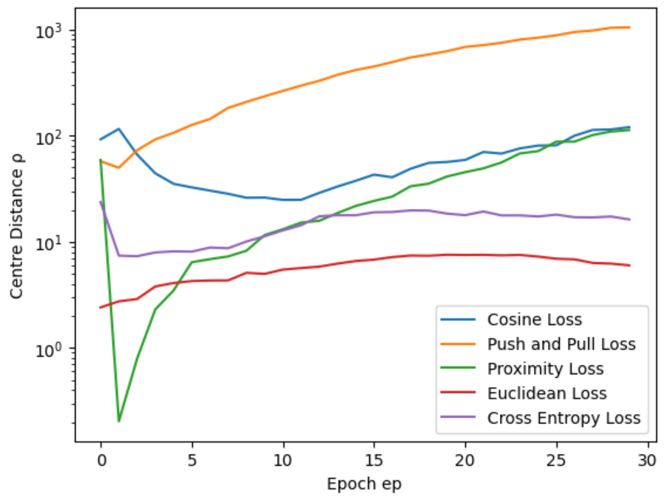


Fig. 34: CIFAR100

D. Centre distance compared to cluster size

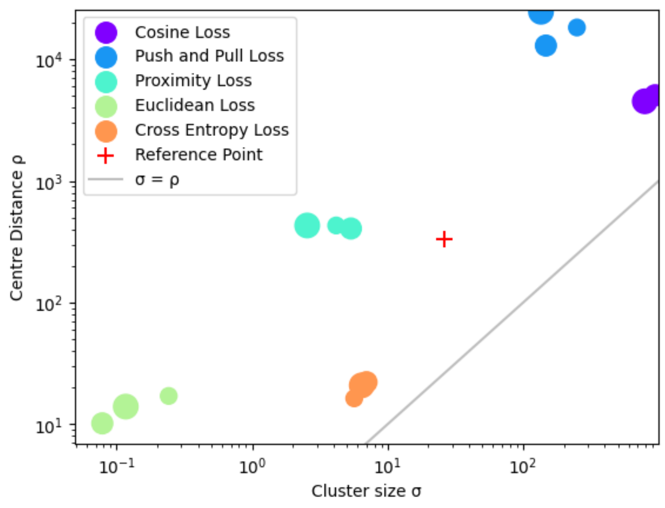


Fig. 35: MNIST

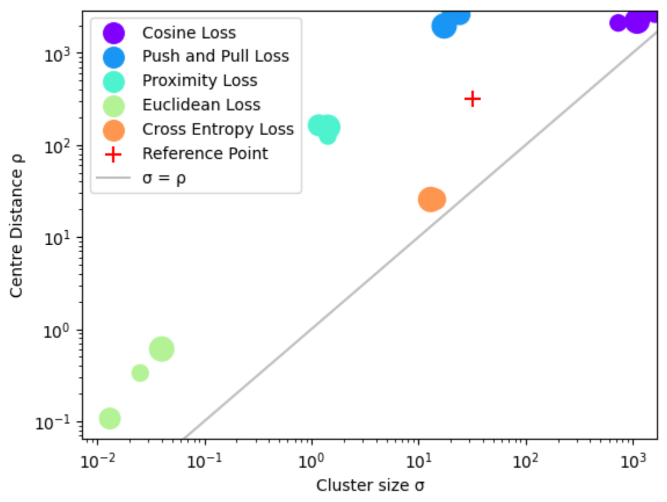


Fig. 36: K-MNIST

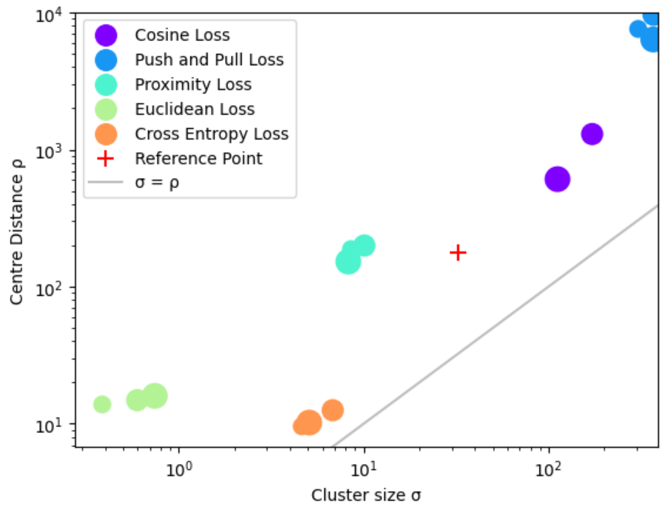


Fig. 37: F-MNIST

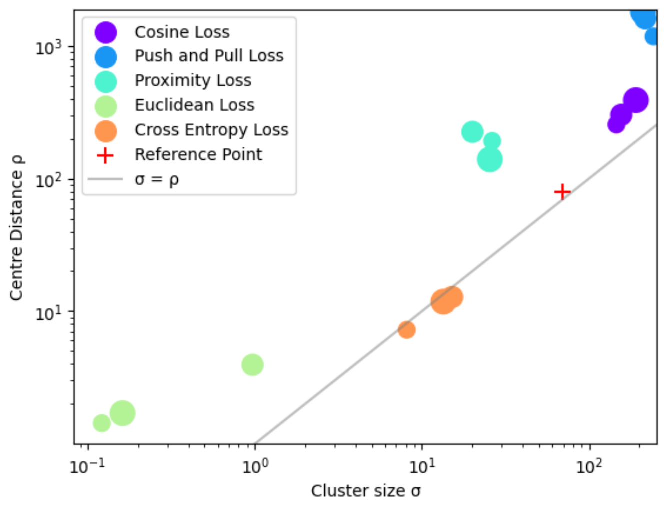


Fig. 38: CIFAR10

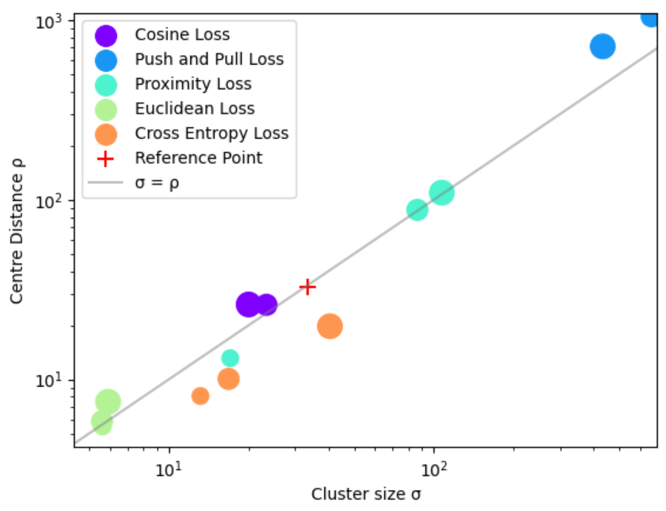


Fig. 39: CIFAR100

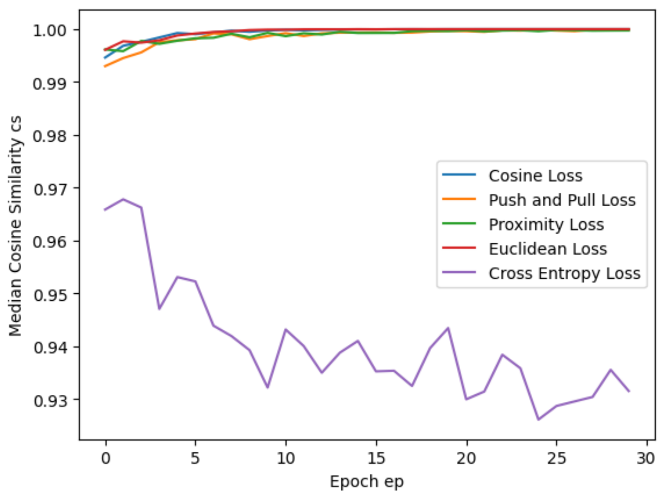


Fig. 42: F-MNIST

E. Median cosine similarity over time

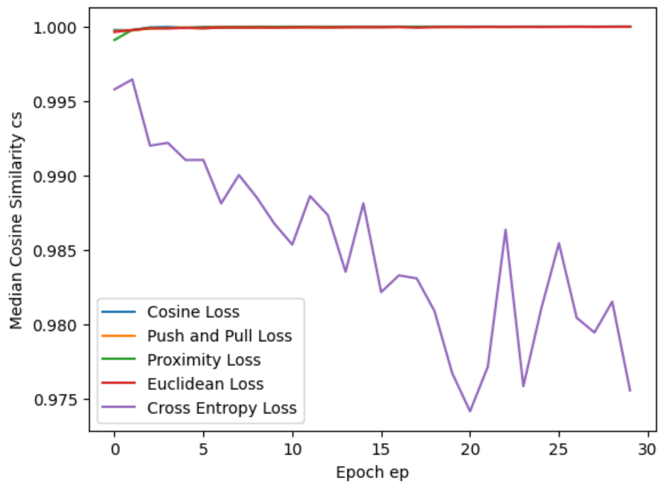


Fig. 40: MNIST

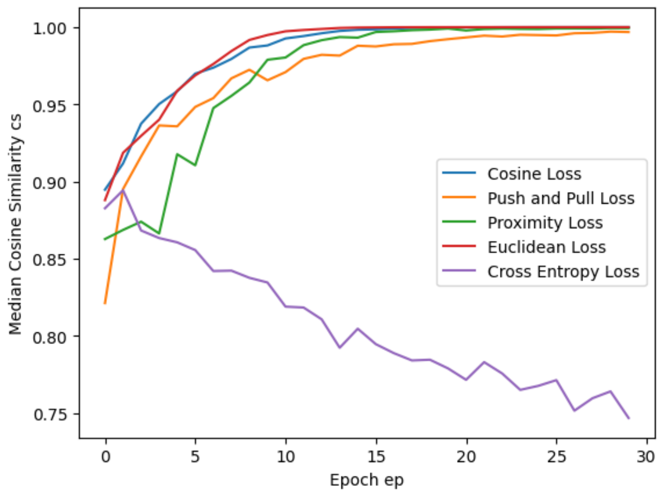


Fig. 43: CIFAR10

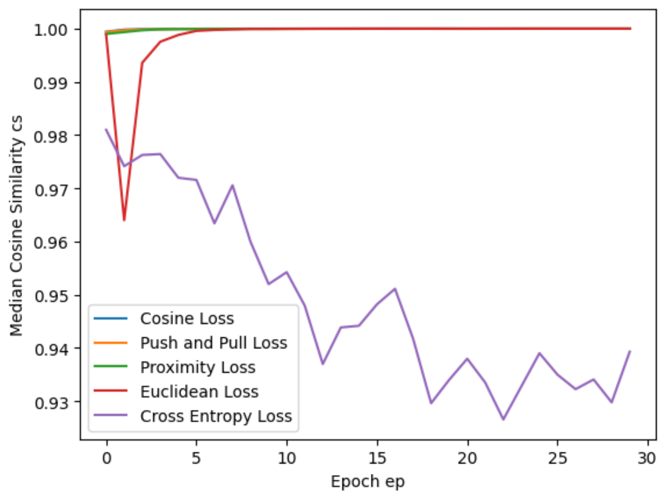


Fig. 41: K-MNIST

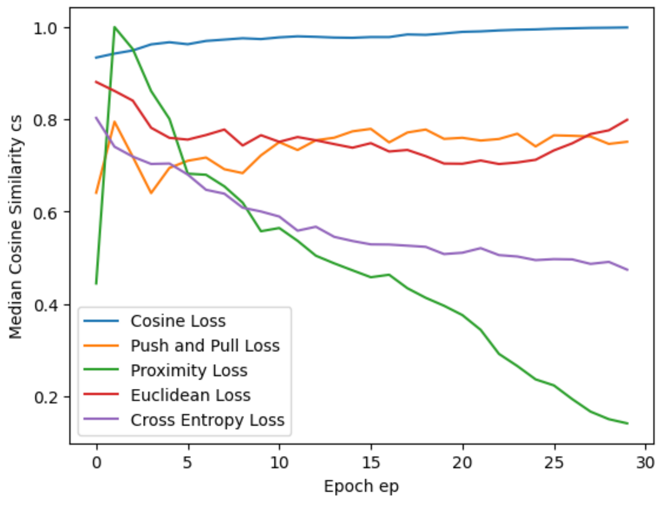


Fig. 44: CIFAR100

F. Cluster size over time

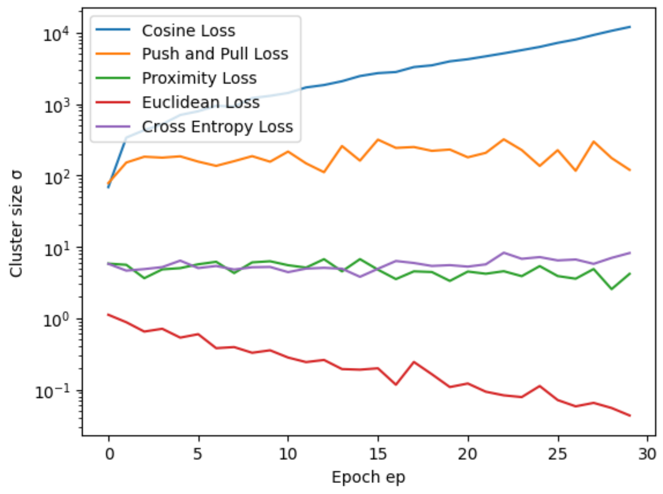


Fig. 45: MNIST

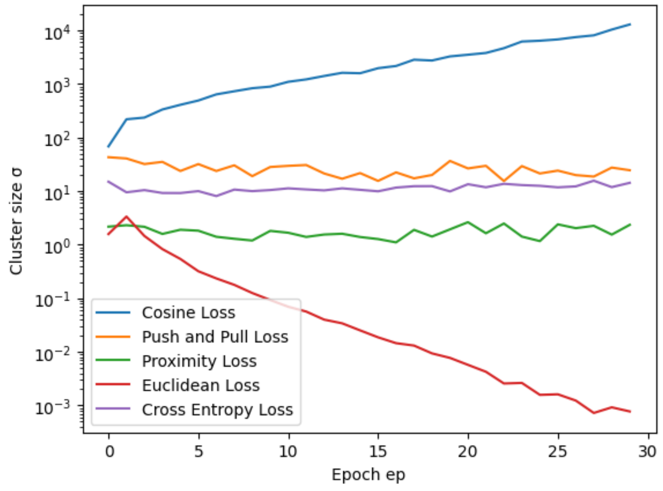


Fig. 46: K-MNIST

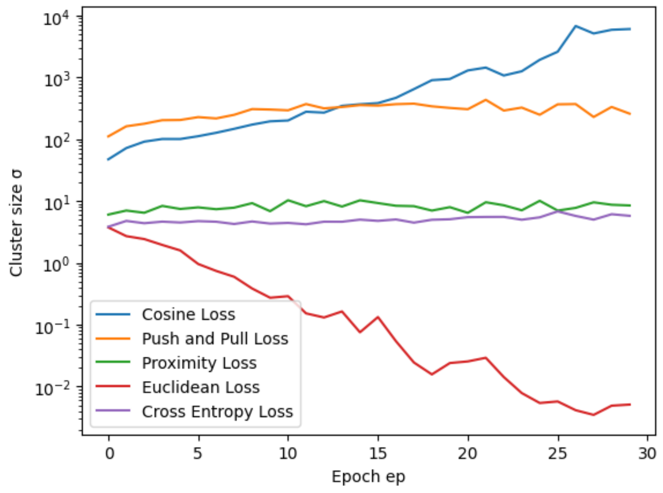


Fig. 47: F-MNIST

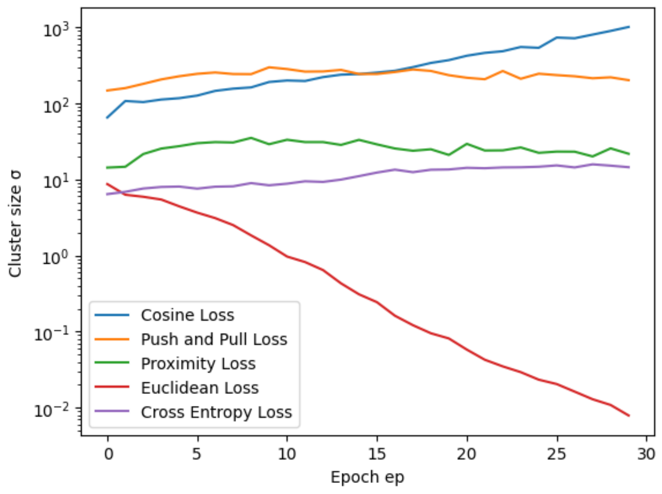


Fig. 48: CIFAR10

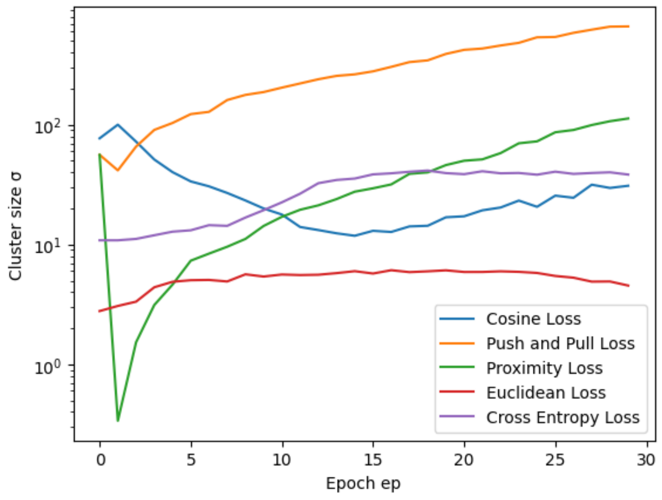


Fig. 49: CIFAR100