UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and Machine Intelligence 2022-2023

Module Coursework Feedback

Module Title: Computer Vision

Module Code: MLMI12

Candidate Number: K5002

Coursework Number: Project

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

	Distinction			**Pass**			**Fail (C+ - marginal fail)**		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

MLMI12 - Contrastive Learning Project

K5002

January 12, 2023

Abstract

In this paper a novel contrastive loss function is presented, which can learn from contrasting image labels, the similarity and differences between images with 78.07% accuracy. The DEIT-B architecture is used and classification performance within 2% of the current state of the art on the TinyImageNet dataset has been verified. Our loss function is a supervised batched pairwise contrastive loss (SBPC), as computing the loss over the entire batch is far more computationally efficient. Results have been ratified during training and through a T-SNE visualisation of the produced embeddings. We conclude that supervised contrastive learning is an effective technique to learn from contrasting image labels, and demonstrate excellent generalisation performance to unseen datasets. However, note that models trained using this technique performed worse than those trained directly for classification...

Contents

1 Introduction

Contrastive learning is the process of training a model to identify similarities and differences between pairs of images. It is especially powerful as a tool for unsupervised pretraining from large image corpora where a network can be trained to recognise augmented versions of the same image. However, it is also useful in the hypothetical supervised setting where the similarity of the subject of various distinct images is known, but not their class; this is the setting in which our investigation takes place. The current state-of-the-art model for the TinyImageNet dataset is the DEIT model[11] which is a distilled version of the revolutionary image transformer[3]. Our implementation of DEIT follows the advancements made by Huynh[4] as well as using a contrastive loss function inspired by one proposed by Pranny et al.[5] which has even been shown to outperform classical multi-class classification learning.

We train the DEIT model under a Siamese setting where the same model is used for all input images and differences in it's output are measured. Since the contrastive learning technique is mostly defined by the loss function, this will be discussed first...

2 Literature Review

2.1 Loss functions

The TinyImageNet dataset consists of a set of 200 classes of images each of which has 500 examples, forming a corpus of 10,000 labelled images. In our training scenario, we will only consider whether pairs of images have the same label or not. Under this paradigm rather than the network producing a class label via a one-hot encoding, it instead produces an embedding. If the images classes are the same, the embedding should be similar; if they are different, the embeddings should be different. Under an optimal scenario, the embedding should tend to a one-hot encoding over classes, however, practically this is very difficult. A *Contrastive Loss* can be used to train a model with a desirable embedding.

The contrastive loss function[1] has the following form[10][2]:

$$\mathcal{L}(\hat{y}_A, \hat{y}_B) = \mathbb{I}(y_A == y_B) \log p(\hat{y}_A, \hat{y}_B) + \mathbb{I}(y_A \neq y_B)(1 - \log p(\hat{y}_A, \hat{y}_B))$$

Note that this is the form of the BCE loss function (Binary Cross Entropy). In this contrastive learning paradigm the identity functions $\mathbb{I}$ denote the cases where A and B are of the same class and of a different class, respectively. While the probability distribution $p(\hat{y}_A, \hat{y}_B)$ is some contrastive measure, commonly being a normalised distance metric between the class predictions; notably $p(\hat{y}_A, \hat{y}_B)$ will have a form similar to the cross-entropy loss thanks to the normalisation. In effect, the contrastive loss function aims to decrease the distance between similar images and increase the distance between dissimilar images. In the case of an imbalanced class, balanced training can be enforced through the use of the triplet contrastive loss function[9], which considers triples (s_a, s_+, s_-), being the anchor, positive and negative samples. A positive image has the same label as the anchor, while negatives have different labels.

$$\mathcal{L}(y_a, y_+, y_-) = \log p(y_a, y_+) + (1 - \log p(y_a, y_-))$$

This can also be extended to consider 1 positive class and $N - 1$ negative classes via the N-pair Contrastive Loss.

The aforementioned methods are suitable for both supervised and semi-supervised **contrastive learning** frameworks. Under the semi-supervised paradigm the true class labels of images are unknown; instead only their similarity is known. Classically this is achieved in an unsupervised manner where the positive example is an augmented version of the anchor, hence the aforementioned methods use of a single positive class. Since the class labels of our images are known we can restrict ourselves to supervised contrastive loss functions and unlock greater performance by considering many positive examples. Since positive and negative examples can be considered from across an entire batch far more comparisons per model prediction can be performed, making training much more efficient.

2.1.1 Supervised Contrastive Losses

Pranny et al.[5] initially proposes a semi-supervised contrastive loss function with similar characteristics to the N-pair contrastive loss, they define it as:

$$-\sum_{i\in I} \log \frac{\exp\left(z_i \cdot z_{j(i)}/\tau\right)}{\sum_{a\in A(i)} \exp\left(z_i \cdot z_a/\tau\right)} \tag{1}$$

Where the denominator aims to be maximised, such that the anchor z_i is similar to some positive $z_{j(i)}$. And the numerator aims to be minimised such that the anchor is different to a number of negative samples. It then extends this with a supervised contrastive loss function:

$$\sum_{i\in I} \frac{-1}{|p(i)|} \sum_{p\in P(i)} \frac{\exp\left(z_i \cdot z_p/\tau\right)}{\sum_{a\in A(i)} \exp\left(z_i \cdot z_a/\tau\right)} \tag{2}$$

Here, the addition of the sum $\sum_{p\in P(i)}$ allows an arbitrary number of positive examples to be considered by using image labels and sampling images of the same class. Hence it is possible to train embeddings that are similar for images of the same class and dissimilar for images of different classes. However, in their research extremely large batch sizes of the order of 2048 were used. This is required for there to be a reasonable probability that a good mix of positive and negative examples from each class will be present, since images across the 200 classes were uniformly sampled. In our experiments only a batch size of 64 was possible, hence in the best case, the size of the set of positives, $|P(i)|$, is likely to be small; $E(|P(i|Bs = 64)|) = \frac{63}{200} = 0.32$. Whereas a 2048 batch size would increase this to $E(|P(i|Bs = 2048)|) = \frac{2047}{200} = 10.2$. Accordingly, in our experiments type II errors (where the network assumes examples are dissimilar even though the true label is the same) are strongly encouraged and the model does not learn. This could be resolved by reducing the number of classes present in each batch via a nonuniform data sampler, or by using a loss that intrinsically accounts for this bias.

2.1.2 SBPC Loss

We devise just such a loss, one that is flexible to any number of positive or negative contrastive pairs. It also best leverages the supervised dataset by comparing all combinations of predictions from a given batch. If the labels of a given pair are equal the **distance** between their embeddings should be minimised and if the labels are different the **similarity** between the embeddings should be minimised. This maximises the utility of each prediction since each is optimised with respect to every other prediction. This approach is more efficient than that proposed by Pranny et al.[5], as it considers a total of $BC2$ pairs in each batch of size B, while Pranny et al. consider exponentially fewer $(B \div 2)C2$ within the same batch size. Our supervised batch pairwise contrastive (SBPC) loss function is defined accordingly.

$$\mathcal{L}_{sbpc} = \sum_i \sum_j \mathbb{I}(y_i = y_j)|\hat{y}_i - \hat{y}_j|^2 + \gamma \mathbb{I}(y_i \neq y_j) \frac{|\hat{y}_i \cdot \hat{y}_j|}{|\hat{y}_i||\hat{y}_j|} \tag{3}$$

Where γ accounts for class imbalance and can be calculated analytically using the ratio:

$$\gamma = \frac{\sum_i \sum_j \mathbb{I}(y_i = y_j)}{\sum_i \sum_j \mathbb{I}(y_i \neq y_j)} \tag{4}$$

This loss function had to be implemented in torch ensuring the differentiability of tensors was preserved; this code listing can be found in appendix figure 9. This loss function is also far easier to implement compared to more classical triplet and N-Pair contrastive loss functions, as it requires no changes to the data-loader compared to training a model for classification. Yet it still conforms to the contrastive learning format as the loss only depends on the similarity between predictions depending on their class.

We also tested a similar loss function (Equation 5) that solely considers distributions over pairwise distances (following the triplet loss function[9]). As well as one that solely considers the similarity between pairs (Equation 6), following the example set by Pranny et al.[5].

$$\mathcal{L}_{dist} = \sum_i \sum_j \mathbb{I}(y_i = y_j)\frac{|\hat{y}_i - \hat{y}_j|^2}{\alpha_i} + \gamma \mathbb{I}(y_i \neq y_j)\frac{1 - |\hat{y}_i - \hat{y}_j|^2}{\alpha_i}, \qquad \alpha_i = \sum_j |\hat{y}_i - \hat{y}_j|^2 \tag{5}$$

$$\mathcal{L}_{sim} = \sum_i \sum_j \mathbb{I}(y_i = y_j)\frac{|\hat{y}_i \cdot \hat{y}_j|}{\alpha_i} + \gamma \mathbb{I}(y_i \neq y_j)\frac{1 - |\hat{y}_i \cdot \hat{y}_j|}{\alpha_i}, \qquad \alpha_i = \sum_j |\hat{y}_i \cdot \hat{y}_j| \tag{6}$$

Both are computationally cheaper as only a single distance/similarity matrix need be computed; however, thanks to the normalisation term a_i, they are less accurate as each prediction within a given batch is assigned equal responsibility for the loss regardless of it's relative error.

2.2 Data

The Tiny ImageNet Challenge[6] is a down-sampled version of ImageNet, it was created in order to reduce the *prohibitively* high training costs of ImageNet. It consists of a total of 10,000 training images over 200 classes (500 training images per class, as well as 50 test and 50 val images). Each image is down-sampled from (on average) 469x387 pixels used in ImageNet to 64x64 pixels, making the dataset smaller but also increasing the difficulty of the challenge (in an attempt to reduce the saturation of ImageNet). The main feature of Tiny ImageNet is that conclusions made on the dataset can be transferred to ImageNet while being upto 100x cheaper to obtain[6].

2.2.1 Augmentation

If self-similarity is used in contrastive learning the augmentation techniques used play a huge role in the accuracy achieved. Accordingly, the augmentation methods used in two state of the art TinyImageNet implementations[4][12] as well as the state of the art contrastive learning implementation[5] was combined to form the image augmentation used in this project.

DEIT[4]	DEIT+PUGD[12]	Supervised Contrastive[5]
Mixup	Random Crop	Random Resized Crop
CutMix	Rand Horizontal Flip	Rand Horizontal Flip
Random Erasing	Normalisation	Colour Jitter
Label Smoothing		Rand Grayscaling
Normalisation		

Table 1: State of the Art Augmentation methods

Pranny et al.[5] mention that Mixup & Cutmix "could potentially improve results further" accordingly we added these augmentation strategies to our ensemble to maximise performance.

Mixup[14] is an augmentation strategy in which images are randomly combined via some mixing ratio α, producing a new image $I_{mixup} = \alpha I_1 + (1 - \alpha) I_2$ where α is a tune-able parameter denoting how much impact the mixed image will have on the original ([4] found $\alpha = 0.8$ to be reasonable). The same operation is also reflected on the image labels, creating a more meaningful version of the commonly used label smoothing algorithm.

Cutmix is similar to Mixup but only mixes local regions of images within an original image, this has the effect of adding distracting objects to training images, which better reflects real-world scenarios where multiple objects of different classes could be present within a scene.

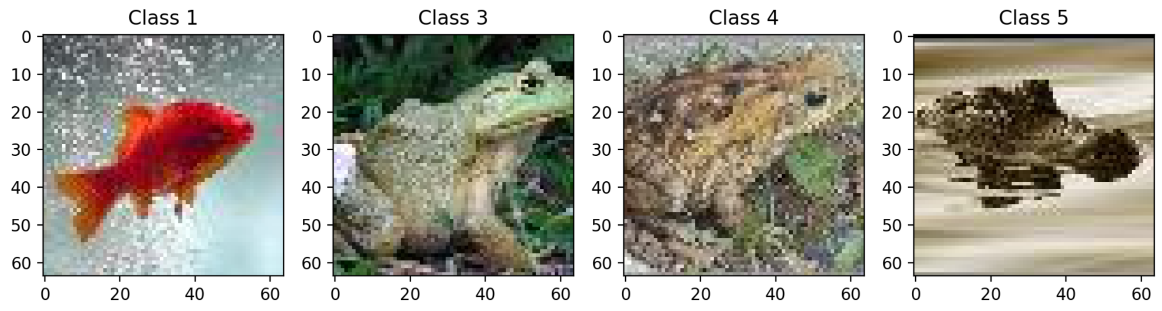


Figure 1: TinyImageNet images

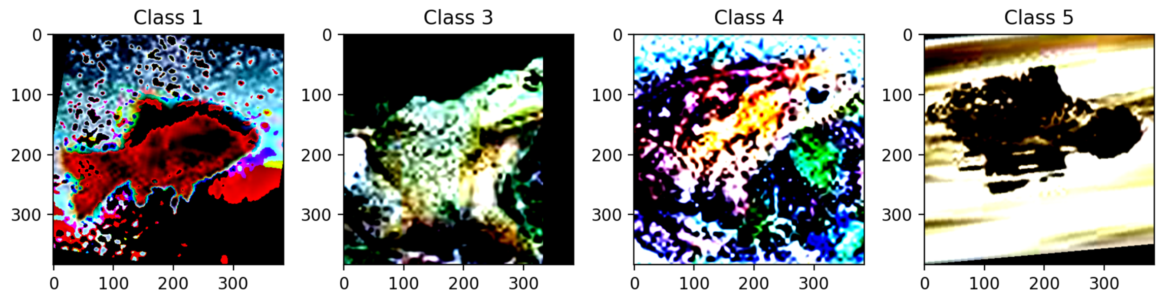


Figure 2: Augmented TinyImageNet images

Figure 1 shows 4 example images from 4 classes, goldfish, frogs, toads and alligators. Combined with the low resolution, the class of the images is difficult to distinguish even for a human. Following augmentation, it becomes even more challenging 2, mostly due to the colour jitter augmentation which alters the contrast and saturation.

2.3 Architecture

The DEIT architecture was chosen as it appears in 4 of the 6 top performing architectures on the TinyImageNet dataset (Table 2).

Architecture	Accuracy
DeiT-B/16-D + OCD[8]	92%
Swin-L[4]	91.35%
DeiT-B/16 + PUGD[12]	91.02%
ViT-B.16 + PUGD[12]	90.74%
DeiT-B/16-D[4]	87.29%

Table 2: State of the art architectures on TinyImageNet

The DEIT[11] architecture is a distilled version of the popular ViT vision transformer proposed by Dosovitskiym et al.[3]. Knowledge distillation is a process that reduces the number of tune-able parameters in a network by using a teacher-student architecture where the teacher in this case is the original trained ViT architecture and the student is a pruned version of the ViT architecture. The student is then trained solely to replicate the classifications produced by the teacher network without any exposure to the teacher networks training data. Thanks to this, although DEIT may be large with 87M parameters, it is far smaller than it's parent, ViT with it's 304M parameters and Swin with 196M parameters. This lower complexity and reduced training time, as well as memory requirement played a key role in picking DEIT to model TinyImageNet. Another small model is ResNet50 with 26M parameters, it would be even faster than DEIT, however it is an older convolutional architecture so only achieves at most 72.39% accuracy[7].

2.3.1 Self-Attention

DEIT and it's parent architecture ViT are transformer architectures, making use of self-attention layers at their heart. The transformer architecture, with it's self-attention layers, was originally developed for natural language processing and was first introduced by Ashish et al. in [13]. It defines self-attention layers in the following way:

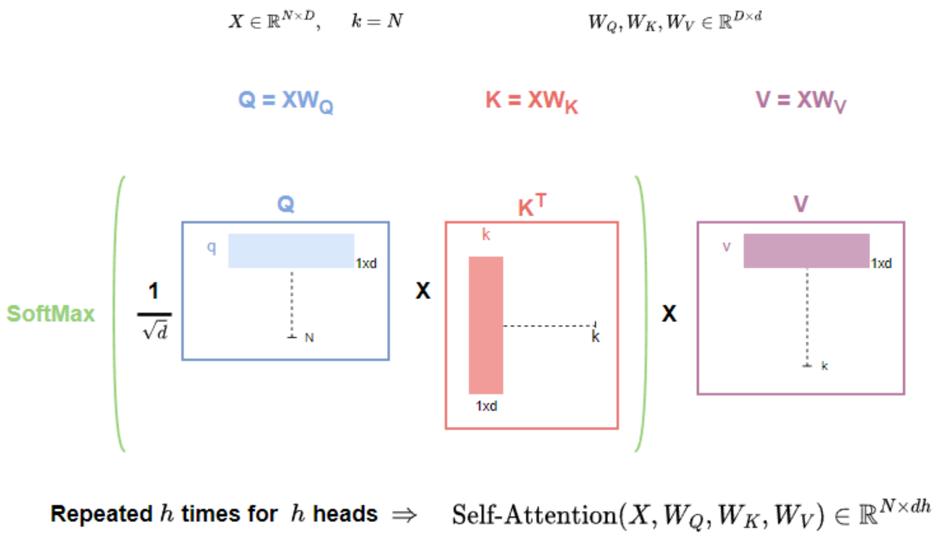

$X \in \mathbb{R}^{N\times D}, \quad k = N$
$W_Q, W_K, W_V \in \mathbb{R}^{D\times d}$
$Q = XW_Q$
$K = XW_K$
$V = XW_V$
SoftMax
$\frac{1}{\sqrt{d}}$
Q
q
1xd
N
X
K^T
k
1xd
k
X
V
v
1xd
k
Repeated h times for h heads $\Rightarrow$ Self-Attention$(X, W_Q, W_K, W_V) \in \mathbb{R}^{N\times dh}$

Figure 3: Self Attention Layer

There are 3 matrices **Query, Key and Value**, each of which are derived from the inputs via a corresponding weight matrix. Then a score is generated by multiplying the query matrix by the transpose of the key matrix. In [13] this operation is considered in terms of the N constituent vectors in Q and K, each of which corresponds to the i^{th} input token.

$$Score_j = \sum_{i \in n} Q_i \cdot K_j \tag{7}$$

Each of the j score vectors is named an *attention head*, since when stacked, self-attention layers give varying *attention* to each of these *attention head* vectors. Next the score is normalised and converted into a probability distribution via the SoftMax function.

$$Prob_j = -\log\left(\frac{e^{Score_j}}{\sum_k e^{Score_k}}\right) \tag{8}$$

Finally, the *value* matrix is applied to this probability matrix, which yields the self-attention context vector.

$$Context_j = Prob_j \cdot Value_j \tag{9}$$

Typically, the self-attention layer is concluded with a linear projection layer to reshape the outputs as desired. The ViT and DEIT architectures follow this trend, but also add dropout layers after both the self attention and projection layers to reduce overfitting.

The ViT architecture[3] was the first realisation that attention based transformer architectures designed for natural language processing could achieve state-of-the-art results. It was derived almost entirely from the highly successful BERT architecture, and consistently outperformed ResNet in their tests (the state of the art deep CNN architectures in use at the time). ViT achieved 88.55% accuracy on ImageNet (A superset of TinyImageNet with more classes and training data).

3 Method

3.1 Preprocessing

In order to decrease loading times while developing the model, the data set was prepossessed from its segmented form, split across thousands of jpeg image files into a single pandas data frame containing torch tensors, which was then compressed into a *pickle* data file. Further, to enable easier development in future all data preprocessing was automated from downloading, extracting and processing the dataset. Notably, some images were stored in greyscale, so had to be up-scaled using openCV's colour converter. These images could have been discarded, but represent real world possible images, so were kept.

Additionally the dataset was divided into the 3 folds at this time, training, test and validation, according to the default splits in the original dataset. The training data was used to train the model, the test data was used to tune hyper-parameters and architecture and the validation data was finally used to test the performance of the final model.

3.2 Data Loader & Augmentation

The dataloader provides a model with a constant supply of data to train with. To maximise performance the dataloader is parallelised across multiple cores of the machine (8) and preemptively processes data before it is requested. A batch size of 128 was used throughout testing, accordingly each batch randomly samples 128 images and labels from the dataset (without replacement) and augments them according to Section 2.2.1. Since the augmentations being performed are complex, this parallelisation is crucial to supply enough data in time, and without it training times double.

3.3 Normalisation

Next it is crucial that the inputs to the network are normalised to have zero mean and unit standard deviation. This is a very intensive process as it requires the entire dataset to be augmented and stored, before their statistics can be measured. Since there are so many images, image arrays had to be incrementally concatenated to reduce storage overheads otherwise, even with a 64GB machine, all memory would be consumed. Ultimately, the means and deviations in the 3 dimensions of the images were calculated to be $\mu = (0.4358, 0.4139, 0.3750)$, $\sigma = (0.2483, 0.2429, 0.2387)$. Computing this alone improved performance by 2% compared to the defaults used for DEIT-B, surpassing the reference model[4].

3.4 Model + Optimisers

Since this paper is investigating the performance of contrastive learning as opposed to architectures and their training to classify images, a pre-trained DEIT-B model was loaded to help reduce training times. Specifically, one trained on a superset of TinyImageNet, ImageNet-1k, in DEIT's original paper[11]. The optimiser is responsible for adjusting training hyper-parameters during training, most notably the learning rate, depending on the loss of the network over time. The default optimiser for most transformer architectures is AdamW[13][11][4] which combines weight decay with the popular Adam optimiser since it was found to be beneficial to BERT[13]. Adam itself is hugely popular as it combines stochastic gradient decent (the most basic form of optimisation) as well as the 1^{st} and 2^{nd} loss moments. Following the leading TinyImageNet DEIT implementation[4], the AdamW optimiser has been combined with cosine annealing. Cosine Annealing anneals the network by regularly *increasing the temperature* by increasing the learning rate, before gradually *decreasing the temperature* with the intention of escaping local optima. However, this does prevent the network from fully converging on the training data, so is disabled after 3 epochs. In order to decrease training time, and because a pre-trained model was used, the first transformer layers were not modified during training; instead, only the following dense layers were modified for classification.

3.5 Embedding Verification

The SBPC loss yield's a network that produces an embedding given an image. These embeddings should form clusters for each image class, which in turn should be linearly separable. To verify this, during evaluation, an SVM was trained using the training data to classify embeddings given the class labels corresponding to the input that produced each embedding. Then this SVM was tested on the same data on which it was trained. Since the SVM only has 200 parameters, it can only perform a very simple linear separation of the data. Accordingly if the embedding is truly linearly separable, the SVM should achieve 100% accuracy when tested on the training data. After training using the SBPC loss, this SVM achieved 99.8% accuracy; therefore, we can validate that the loss is functioning correctly. Interestingly, this did not hold for the two additional losses proposed in Equations 5 and 6, with each only being able to map from embeddings to training labels with an SVM correctly 74.19% and 79.96% of the time respectively. The results have been summarised in Table 3. The distance and similarity only loss functions were discarded at this time as this test shows that they could not even fit the training data correctly.

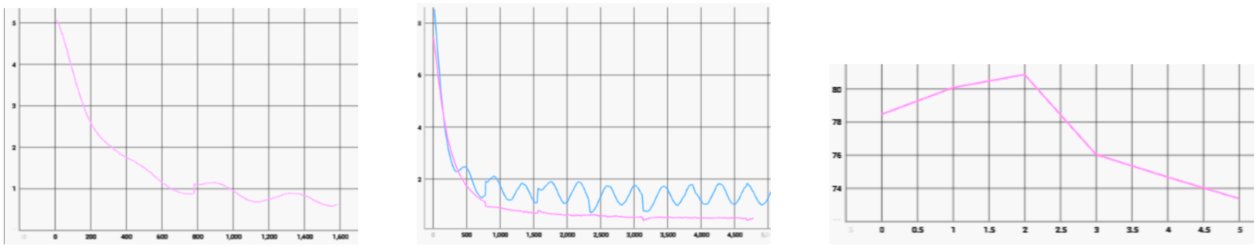
To measure the accuracy of the model, this same SVM was then tested on the test data, and it's accuracy was measured. Accuracy was measured both on top class, where the class with the greatest probability is taken and compared to the true label. And also on the top 5 classes, where any of the top 5 most probable classes are compared to the true label. The KNN (k-nearest neighbour) algorithm was also tested as a substitution for the SVM, but since the embedding clusters are not perfectly globular the KNN was not sufficiently powerful to classify the data. Typically, maximum accuracy was achieved after training for 1 to 3 epochs, this was to be expected as only a relatively small number of parameters had to be tuned.

Loss	Embedding fit to training data	Eqn
SBPC	**99.8%**	3
SBPC Dist only	74.19%	5
SBPC Sim only	79.96%	6

Table 3: Training Embedding Linear Separability

3.6 Metrics

Training metrics were recorded using tensorboard in order to verify model convergence, the most important of which is the training loss. For example, the final model's loss curve has been shown in 4a. After 800 batches (corresponding to one epoch) the model converges to it's final loss, the loss fluctuates due to the cosine annealing used but barely decreases. To verify that cosine annealing was not reducing model accuracy, 2 models were trained for 7 epochs, in one cosine annealing was disabled after 1 epoch. Figure 4b shows that this does allow the loss to converge to a lower value; however, the accuracy achieved by disabling Cosine Annealing was less than it's counterpart 75.83% vs 78.07%. Figure 4c demonstrates the optimal number of epochs to train a model for, before it overfits the training data and test accuracy decreases. Although the results relate to models trained for classification, it is reasonable to assume that they will transfer well to the contrastive learning paradigm, as a high classification score corresponds to a high degree of image understanding encoded within the model.



(a) Best Model (b) annealing (c) Over-fitting

Figure 4: Tensorboard metrics

4 Results

Theoretically any supervised contrastive learning algorithm should learn at least as effectively as a non-contrastive algorithm since the same training data is accessible in either case. A good comparison will be provided by comparing a similar implementation for TinyImageNet from [4] and our own implementation on DEIT, both of which will be trained for image classification in a non-contrastive manner (Cross Entropy Loss).

Description	Accuracy	Top 5 Accuracy
Reference Cross-Entropy[4]	87.29	95.52
Cross Entropy	89.94	96.09
SBPC	78.07	91.65
SBPC (CIFAR)	73.91	93.29

Table 4: Results

The model trained using the cross-entropy loss performs even better than the reference implementation used by Huynh[4], this is due to 2 key improvements. Firstly the data-loader normalisation values were incorrect for the reference implementation's chosen augmentations, whereas we computed normalisation values for our chosen augmentations according to Section 3.3. Secondly we combined the augmentation methods used by Huynh[4] and Pranny et al. [5] which were found to improve accuracy.

Despite the contrastive learning paradigm adding significant complexity to the model and it's loss function, the obtained results were very impressive. When trained using a contrastive loss, the 200 element embedding produced by the model encodes the classes of each sample in a 200D space that is linearly separable. To map this high-dimensional space to a single class a simple SVM was used to adapt the embedding into a classification model which was trained on the class labels for the corresponding embedded images. This can be used without diluting the significance of the results because an SVM is a very simple model that can only identify very basic structures which our architecture has already extracted.

Testing this model on the evaluation split yielded 78% accuracy at correctly identifying the similarities and differences of images using the DEIT architecture with the SBPC loss. Our model is sub-optimal as it should be able to achieve the same accuracy as the DEIT architecture trained for classification on the same dataset using the cross-entropy loss function which scored 89.94% accuracy. As such our model is 12% worse than theoretically possible. Interestingly, the performance disparity when considering the top 5 predictions of the model was just 4.4% showing that the embedding has successfully captured much of the information in the classes of the images via the SBPC loss function. This can be visualised by performing T-SNE dimensionality reduction to the embeddings produced by the network to reduce the 200 dimensional embedding to a 2 dimensional representation.

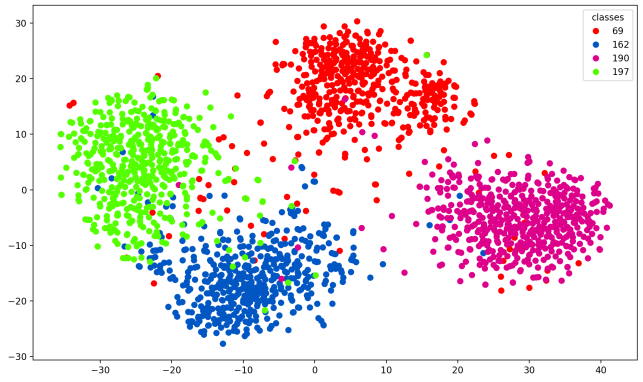


Figure 5: T-SNE dimensionality reduction of SBPC embeddings

4 classes were randomly chosen to display in a scatter graph (Figure 5). You can clearly see that the 200-dimensional embedding has the desired behaviour of maximising the distance between dissimilar images and minimising the distance between similar images, since each class forms a distinct globular cluster. Accordingly, the SBPC loss function has trained a desirable embedding that recognises the similarity between images. By visualising some inliers and outliers from class 162, some of the errors in the model can be explained. Sampling from the center of the cluster yields images that are highly characteristic of the class of houses, all of which show greenery and a single building as the subject of the image. Notably, the first two outliers had very similar embedding representations, both appearing at (-20,15) on the T-SNE reduction, therefore it is not surprising that the subject of both is very similar. Accordingly it seems that the model struggles with tropical, coastal houses, potentially being thrown off by the sea in the image. Finally, outlier 6 is extremely challenging, being a small segment of thatched roofing. So it is understandable to miss-classify outlier 6.

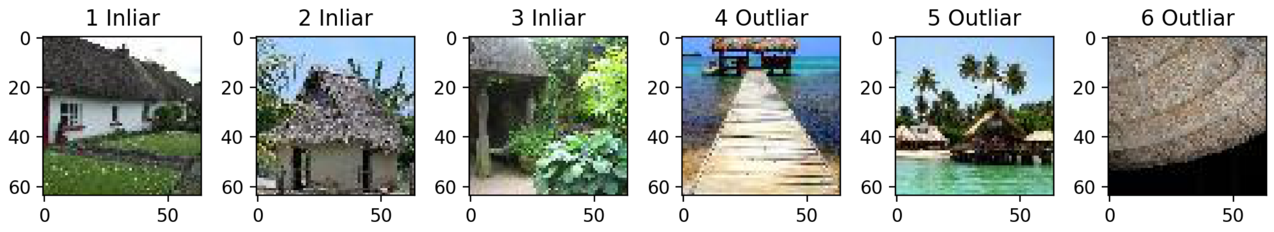


Figure 6: Outliers

4.1 CIFAR Cross Validation

The same model that was trained on TinyImageNet, was also tested on CIFAR-100 following the same testing strategy that was used for TinyImageNet. CIFAR-100 is a distinct dataset versus TinyImageNet with largely different categories, so provides a good test of whether our model correctly differentiates between similar and dissimilar images. Evaluation on this dataset yielded a score of 73.91%(Table 4); which is just 4% worse then our accuracy on TinyImageNet's test set. This proves that our model generalises well to unseen classes and image subjects, so has correctly distilled the ability to detect image similarity. To further verify the seperability of classes in the embedded representation of CIFAR the same T-SNE reduction was performed as before (Figure 7).

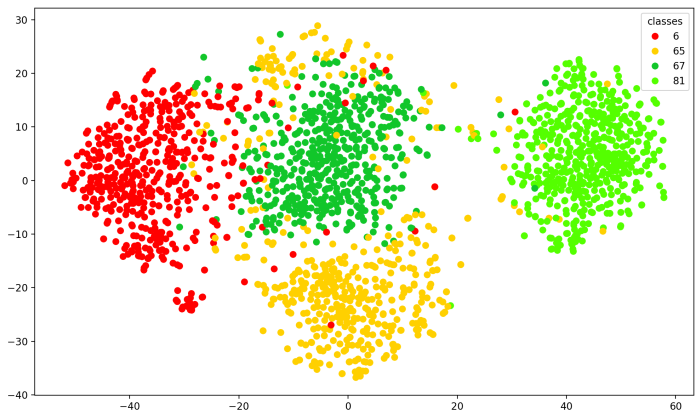


Figure 7: T-SNE dimensionality reduction of SBPC embeddings on CIFAR

The behavior on the CIFAR dataset is very similar to that on TinyImageNet, with the four classes forming distinct globular clusters. However class 65 in particular is notably less distinct from class 81, results such as these could cause the performance disparity between TinyImageNet and CIFAR. Delving further into the outliars (Figure 8) from class 65 reveals that it consists of rabbits and that the poorly classified images are particularly low resolution with respect to the image subject. It would be extremely difficult for even a human to distinguish the rabbits in the images. Furthermore, the rabbit category is absent from the TinyImageNet dataset, which could explain the reduced accuracy on this class. Overall, that testing the model on CIFAR has further demonstrated that the SBPC loss has function correctly in training the model to distinguish between different subjects in an image.

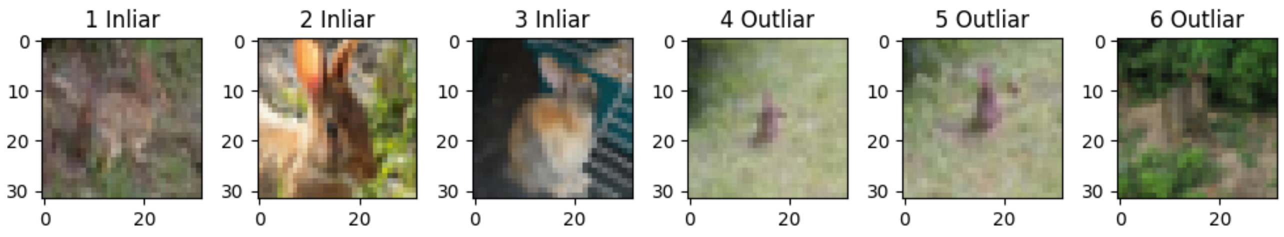


Figure 8: CIFAR outliers

5 Conclusion

In this project, a novel loss function SBPC has been presented, which has been able to learn from contrasting image labels the similarity and differences between images with 78.07% accuracy. In doing so, the current state of the art for TinyImageNet has been developed upon, with performances validated within 2% of the current state of the art implementation using OCD[8]. We have also proven that SBPC performs as expected through extraneous verification during training (Section 3.5), during our T-SNE visualisation (section 4) and through our CIFAR cross validation. Overall, the work done in this project could be extended further toward an optimal contrastive loss function that achieves an accuracy closer to the theoretically attainable 89.94% accuracy.

6 Appendix

```
pdist = nn.PairwiseDistance(p=2)

dist_matrix = pdist(pred, pred.unsqueeze(1)) # pairwise distance matrix
sim_matrix = pairwise_cosine_similarity(pred, pred)

# Remove the main diagonals - ignoring identical pairs
non_diagonal_mask = 1 - torch.eye(dist_matrix.shape[0]).to(device)
dist_matrix = dist_matrix * non_diagonal_mask
sim_matrix = sim_matrix * non_diagonal_mask

uy = y.unsqueeze(1)
positive_mask = torch.eq(uy,  uy.T).float() # all the paris with the same labels are
#            set to 1, otherwise 0

positives = positive_mask * dist_matrix
positive_count = (positive_mask*non_diagonal_mask).sum() # how many positive
#            examples in this batch? (excluding the diagonals)

negated_negative_mask = positive_mask - 1 # all the pairs with different labels are
#            set to -1
negatives = -1 * negated_negative_mask * sim_matrix
negative_count = -1 * negated_negative_mask.sum() # how many negatives in this batch

if torch.min(positive_count, negative_count) == 0: # failsafe in the unlikely event
#            a batch is purely positive or negative
    return 0

positive_negative_balance = negative_count / positive_count # the ratio of negatives
#        to positives, this will later scale down the negatives, for >2 classes,
#        negatives are far more likely in a given batch

loss = positives + (negatives / positive_negative_balance)

batchsize_norm = torch.sqrt(positive_count+negative_count) #normalise by batch size

l = loss.sum() / batchsize_norm # for the overall loss
return l
```

Figure 9: Proposed Supervised Batch Pairwise Contrastive Loss

References

[1] S. Chopra, R. Hadsell, and Y. LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 539–546 vol. 1. DOI: 10.1109/CVPR.2005.202.

[2] *Contrastive Representation Learning — Lil'Log*. https://lilianweng.github.io/posts/2021-05-31-contrastive/. (Accessed on 01/02/2023).

[3] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[4] Ethan Huynh. *Vision Transformers in 2022: An Update on Tiny ImageNet*. 2022. DOI: 10.48550/ARXIV.2205.10660. URL: https://arxiv.org/abs/2205.10660.

[5] Prannay Khosla et al. "Supervised Contrastive Learning". In: *CoRR* abs/2004.11362 (2020). arXiv: 2004.11362. URL: https://arxiv.org/abs/2004.11362.

[6] Ya Le and Xuan S. Yang. "Tiny ImageNet Visual Recognition Challenge". In: 2015.

[7] Zicheng Liu et al. *Decoupled Mixup for Data-efficient Learning*. 2022. DOI: 10.48550/ARXIV.2203.10761. URL: https://arxiv.org/abs/2203.10761.

[8] Shahar Lutati and Lior Wolf. *OCD: Learning to Overfit with Conditional Diffusion Models*. 2022. DOI: 10.48550/ARXIV.2210.00471. URL: https://arxiv.org/abs/2210.00471.

[9] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *CoRR* abs/1503.03832 (2015). arXiv: 1503.03832. URL: http://arxiv.org/abs/1503.03832.

[10] *The Beginner's Guide to Contrastive Learning*. https://www.v7labs.com/blog/contrastive-learning-guide. (Accessed on 01/02/2023).

[11] Hugo Touvron et al. "Training data-efficient image transformers & distillation through attention". In: *CoRR* abs/2012.12877 (2020). arXiv: 2012.12877. URL: https://arxiv.org/abs/2012.12877.

[12] Ching-Hsun Tseng et al. "Update in Unit Gradient". In: *CoRR* abs/2110.00199 (2021). arXiv: 2110.00199. URL: https://arxiv.org/abs/2110.00199.

[13] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[14] Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *CoRR* abs/1710.09412 (2017). arXiv: 1710.09412. URL: http://arxiv.org/abs/1710.09412.