UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and Machine Intelligence 2019-2020

Module Coursework Feedback

Module Title: Computer Vision

Module Code: 4F12 / MLMI12

Candidate Number: H801L

Coursework Number: Mini-project – Deep Learning

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.

Image similarity classification via Deep Metric Learning

Candidate number: H801L, Word count: 4000

26th December 2020

1 Introduction

1.1 Image matching task

In the image matching task, a classification model receives a pair of images and has to determine whether these are similar (represent the same object) or dis-similar. To accomplish this task we leverage (Deep) **Distance Metric Learning** approach [1], where we extract a lower dimensional representation of query images $G_w(\{\boldsymbol{x}, \boldsymbol{x\prime}\})$, such that the distance between those images D_w, will be small if they are similar and large otherwise.

$$D_w = ||G_w(\boldsymbol{x}) - G_w(\boldsymbol{x\prime})|| \tag{1}$$

Effectively, the objective is to learn to cluster together images of a similar semantic meaning in a K-dimensional space, where K is the size of the image embedding vector.

During inference, we would obtain embedding vectors for query images and evaluate their similarity by checking either whether their embedding vectors belong to the same cluster or whether the euclidean distance between embedding vectors is below some predefined threshold (clustering and the threshold can be defined using the validation set).

1.2 Problem analysis & report organisation

Deep Metric Learning can be divided into two main sub problems: identifying objective function and designing a CNN feature extractor.

The choice of objective function is important, as it will determine how the weights of a CNN feature extractor will be adjusted to form the most meaningful embeddings for image clustering purpose. We discuss and evaluate different approaches in Section 2.

Depending on choice of objective function, our training network can take form of a Siamese network (two parallel networks with shared weights), Triplet network (three parallel networks with shared weights), or a lifted structure (a single network evaluating N query images). In each case, we need a powerful encoder which can accurately extract and represent semantic meaning of images. For this purpose we use CNN networks which are effective at learning spatial image features. Our exploration and evaluation of different CNN architectures is presented in Section 3.

In section 4 we present our training procedure and explore techniques for speeding up convergence, such as pair mining. In section 5 we conduct evaluation of our final model.

1.3 Dataset

We train and evaluate our model on 64x64 images from the Tiny ImageNet dataset, comprising 200 image classes with 550 samples per class. We decided to reduce task complexity by considering only first 100 image classes, to reduce the time required for training. We split the datset to: training, validation, and test sets in 45000, 5000, 5000 image proportions.

2 Objective functions

There are numerous objective functions one could use for Deep Metric Learning. For this project we test and evaluate 3 simple ones: binary cross-entropy, contrastive loss [2], and triplet loss [3], [4]. Other, objective functions presented in the literature include: Multiclass N-pair loss [5], lifted structure loss [9], and Angular loss [8]. While we won't test the later loss functions, we decided to mention them briefly, as they address important limitations of contrastive and triplet losses.

Note in the following equations, $y_n = 1$ denotes ground truth for similar pairs.

2.1 Binary cross-entropy based sigmoid

We devised this loss function ourselves. It computes Euclidean distance between embeddings of both images, computes activation a_n, and feeds it to a sigmoid function, such that the sigmoid will determine whether images are similar $\sigma(a_n) \to 1$, or dissimilar $\sigma(a_n) \to 0$. However, because Euclidean distance is strictly-positive we first apply batch normalisation on the input to sigmoid layer. Following the batch normalisation, small euclidean distances will have negative value, therefore we expect the network to learn negative weight w.

$$\mathcal{L} = -\frac{1}{2N}\sum_{n=1}^{N} y_n log(\sigma(a_n)) + (1 - y_n) log(1 - \sigma(a_n)) \tag{2}$$

$$a_n = w \cdot \frac{D_w - \mu(D_w)}{std(D_w)} + b \tag{3}$$

2.2 Contrastive loss

Contrastive loss was proposed by Chopra [2], and uses a Siamese network. It directly minimises square euclidean distance, D_W^2, between similar images and penalises dis-similar pairs for which the distance is smaller than the margin m.

$$\mathcal{L} = \frac{1}{2N}\sum_{n=1}^{N} (y_n) \left(D_W\right)^2 + (1 - y_n) \left(\max\left(0, m - D_W\right)\right)^2 \tag{4}$$

Contrary to Binary cross-entropy, Contrastive loss (and Triplet loss) can't predict directly whether a pair is similar or not. As mentioned in the introduction the objective functions makes the network learn useful embeddings, and the classification can be later done via clustering or by comparing distance against a threshold.

2.3 Triplet loss

To implement triplet loss we need to extend the Siamese network by another CNN feature extractor into a triplet network.

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \max\left(0, \left\|G_w(x_n^A) - G_w(x_n^P)\right\|_2^2 - \left\|G_w(x_n^A) - G_w(x_n^N)\right\|_2^2 + \alpha\right) \tag{5}$$

Each training input consists of three images, an anchor image x_n^A, one positive (similar) image x_n^P, and one negative (dis-similar) image x_n^N. This loss looks at relative distance between positive and negative pairs, forcing the negative pair to have a greater distance then the positive pair plus the constant α.

2.4 Other loss functions

Loss functions presented thus far, look only at a single negative (dis-similar) pair at a time. As a result, at each step, we separate two negative images away from each other, however as a byproduct, we might be pushing them closer to images from other classes. This is one of the reasons why these losses suffer from slow convergence.

Multiclass N-pair loss, addresses this problem by considering M+1 other images at every step. Out of those images, one is a positive pair to the query image, and remaining M are all dis-similar to the query image and, ideally, they are of different classes. This loss is presented in Equation 6.

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\log\left(1+\sum_{i=1}^{M}\exp\left(f^A f_i^N - f^A f^P\right)\right) \tag{6}$$

$$f^A f^P = \left\|G_w(x_n^A) - G_w(x_n^P)\right\| \tag{7}$$

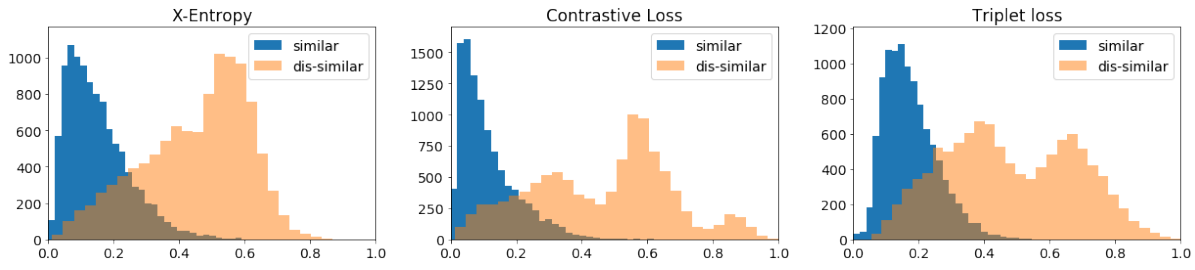
Lifted structure loss aims to utilise all N^2 image pairs in a mini-batch, rather then N pairs as contrastive and triplet losses do. It then tries to minimise distances for all positive pairs and make distances for all negative pairs greater then margin α, which makes it similar to the N-pair loss.

Loss functions don't have to only restrict to evaluation of distances in euclidean space. **Angular loss**, for example, has successfully integrated angle relationship into similarity computation, which resulted in making the loss function more robust to image variances.

2.5 Comparing objective functions

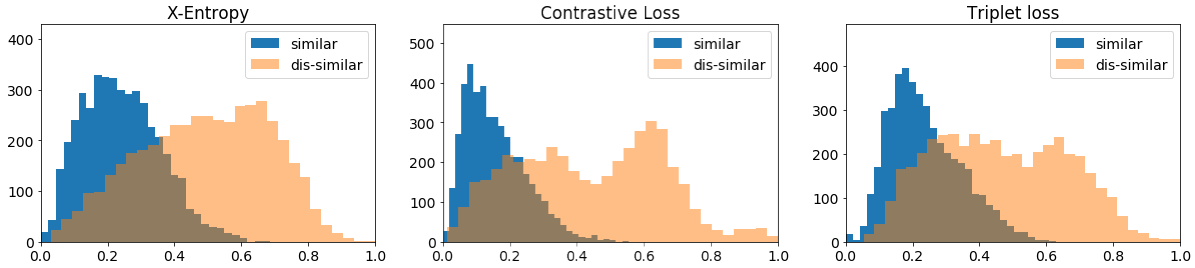
We compare three objective functions using MNIST-digit dataset and a simple LeNet-5 like network for feature extraction. For the dataset, labels $\{2, 5, 8\}$ were removed such that we can also test performance on unseen data.

Figures 1 and 2 show distributions of normalised distances between test image pairs, for seen and unseen categories, respectively. For network with Contrastive loss, the separation between means of distributions for similar and dis-similar pairs was the highest for both seen and unseen validation data, $\mu_{dist} = 0.349$ and $\mu_{dist} = 0.282$ respectively. This network has also achieved highest validation accuracy, on all data, as presented in Figure 3. We have observed that the cross-entropy loss does not offer any advantages over the contrastive loss, and thus will be disregarded at this stage.



(a) $\mu_{dist} = 0.300$ (b) $\mu_{dist} = 0.349$ (c) $\mu_{dist} = 0.305$

Figure 1: Distribution of normalised distances between similar and dissimilar image pairs. Evaluated on test image pairs of SEEN categories.



(a) $\mu_{dist} = 0.249$ (b) $\mu_{dist} = 0.282$ (c) $\mu_{dist} = 0.226$

Figure 2: Distribution of Normalised distances between similar and dissimilar image pairs. Evaluated on test image pairs of UN-SEEN categories.

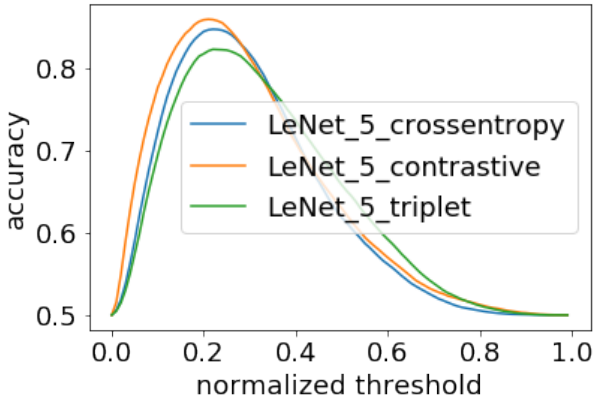


Figure 3: Final test accuracy for networks with different objective functions.

Contrastive and Triplet losses have been widely used in the literature. However, they both suffer from similar limitations. As mentioned earlier, they only look at single pairs or triplets at a time, thus while separating two negative images, we might be unwillingly, moving them closer to images from other classes. While class clusters will be formed eventually, this process is ineffective. Another limitation of these two methods is slow convergence at later stage of training, when the fraction of trivial pairs[1] in the training set increases. This limitation can be overcome via Pair mining which is explored in Section 4. However, we believe that triplet loss is more likely to suffer from this problem then the contrastive loss. Since it looks at a relative distance, we might not only need to mine Hard-negative pairs, but also Hard-positive pairs.

We have also noted that Triplet loss produces clusters which are not as concentrated as those for contrastive loss. Contrastive loss has a direct incentive to reduce distance for positive pairs to zero. Resultant clusters have lower variance and the there is less chance for cluster overlap. Therefore, under limited training time, contrastive loss could prove better at making all clusters clearly distinct.

To visualise this, we run another experiment, this time with data for all digits and using a 2-D embedding vector (for experiments in Figures 1 and 2, we used 48-D vector). Figure 4 illustrates the difference between absolute and relative distance evaluation. Clusters generated by Contrastive loss are much more concentrated, for triplet loss the cluster variance is higher.

Following our discussion, we decide to proceed with **contrastive loss** for this project.

[1]pairs which are already correctly classified and don't contribute significantly to the loss

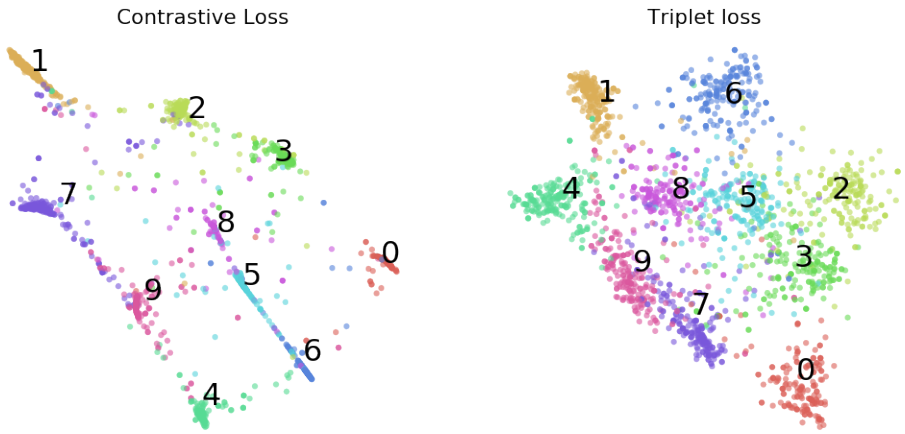


Figure 4: Visualisation of 2D embedding vectors for sample of 2000 test image pairs, for contrastive and triplet losses.

3 CNN architecture

For the feature extractor part of our model we explore VGG and ResNet architectures, which have proved successful during ILSVRC competition. Due to lower image resolution and lower number of classes in the Tiny ImageNet dataset, we believe that minor modifications to original networks will improve performance on our task. The following sections present our experiments and modifications to those networks.

3.1 VGG-16

The original VGG-16 architecture network [11] was used for classification of 256x256 images and 1000 classes. In Tiny ImageNet images are 64x64 in resolution, thus we remove the last Max Pooling layer in order to keep higher channel resolution before the fully connected layers. Further, we hypothesise that, due to reduced number of classes and smaller image resolution, we can reduce the model capacity without much loss on performance. Reducing redundant model capacity may help with generalisation power of the network and improve networks learning ability due to fewer model parameters and less flops required for forward and backward passes.

Experiment description We ran a series of short experiments, lasting 7 epochs and trained on a dataset of 100 classes and 200 samples per class. These will inform us about a more precise range of plausible architecture parameters. Afterwards, we conduct a longer random search experiment over the most promising parameters. This experiment will run for 20 epochs, using 100 classes and 450 samples per class. We use batch size of 64 image pairs and a learning rate of 0.001 with exponential decay by a factor of 0.9 every epoch, which we found to work well empirically.

For each experiment we compare validation loss and validation accuracy evaluated at best checkpoint for a model - we compute euclidean distances between validation image pairs and record model accuracy at different similar/dis-similar separation thresholds. In order to speed up the training procedure we implemented a Data Generator (see Section 4), it improved training convergence and significantly streamlined the experimentation process.

Fully-Connected layers In this experiment we compare models with different number of neurons in the two, final fully-connected layers. We compare 4096, 2048, and 1024 neuron combinations.

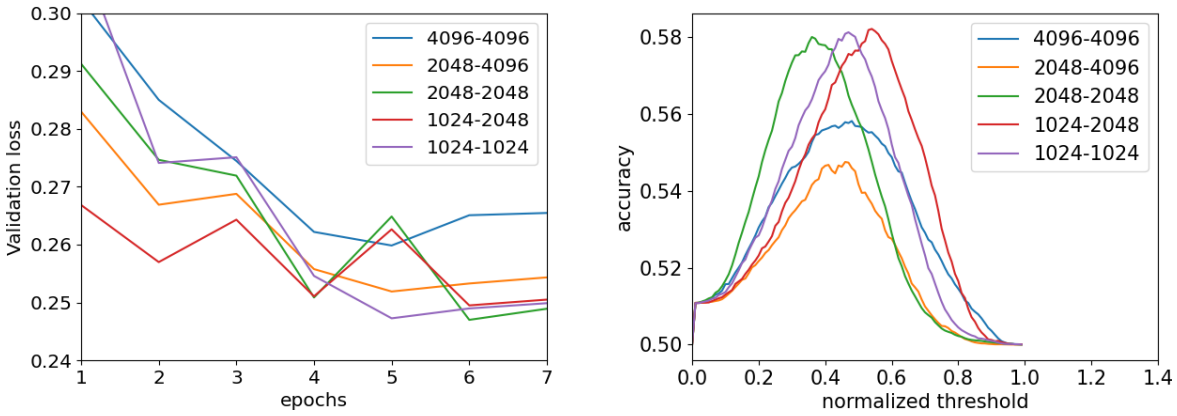


Figure 5: Comparing models with different number of neurons in final fully connected layers ("second to last"-"last layer").

Both plots on Figure 5 shows that models with reduced number of neurons in the final FC layers experienced faster initial training convergence. This could be due to smaller number of neurons to update and thus the ability to form useful features faster. Model with 1024-1024 configuration of neurons in FC layers has reduced the number of neurons in VGG from 65 down to to 24 million.

Final embedding layer VGG-16 network was built for classification, therefore the final softmax layer needs to be replaced with image embedding layer. It is unclear, however, how big embedding vector should be. The search space grows with number of dimensions, therefore as we increase the length of the embedding vector the number of dimensions along which we can try to differentiate different classes increases, which should improve clustering performance. However, as with other layers, the more neurons in the layer, the more high level features can be unique only for a few images in a specific class, and the overfitting may occur.

Figure 6 shows that all embedding lengths are viable options, achieving similar validation losses. Validation accuracy is highest for lengths between 96 and 160 suggesting that these could be optimal values. As guided by observations on MNIST experiments where only a single 2D vector was sufficient to separate categories, we didn't test vector lengths beyond 200, as this seemed unnecessary and could introduce overfitting at later stage of training.

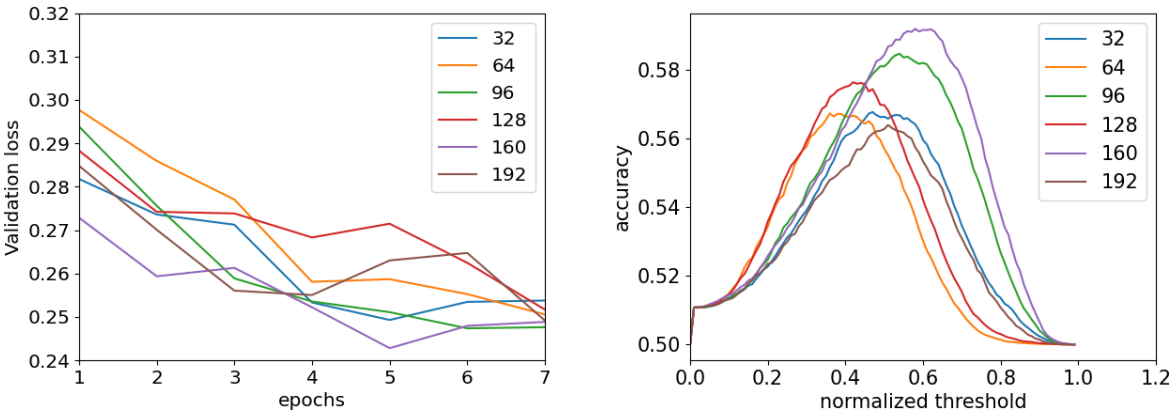


Figure 6: Comparing models with different lengths of embedding vectors.

Depth of Feature maps While the majority of parameters are in the fully connected layers, the most computation happens in the convolutional layers. We believe that due to lower image resolution there will be not as many features to recognise by the first convolutional block. Additionally, because there are less classes, we expect there to be less feature combinations to discover in the later layers, thus their depth can be also reduced. We decided to integrate experiment on depth into the longer random search experiment, presented below. Since number of feature combinations grows with layer we constrain random selection of a layer depth to be at least as large as previous layer and not bigger then 128,256,512,512, for 2^{nd}, 3^{rd}, 4^{th}, and 5^{th} conv. blocks respectively. First block can take values 32 or 64.

Choosing final architecture It is insufficient to analyse hyper-parameters independently, as they may perform differently when in different combinations. To select the best combination we conduct a random search over network parameters configurations, with results presented in Figure 7.

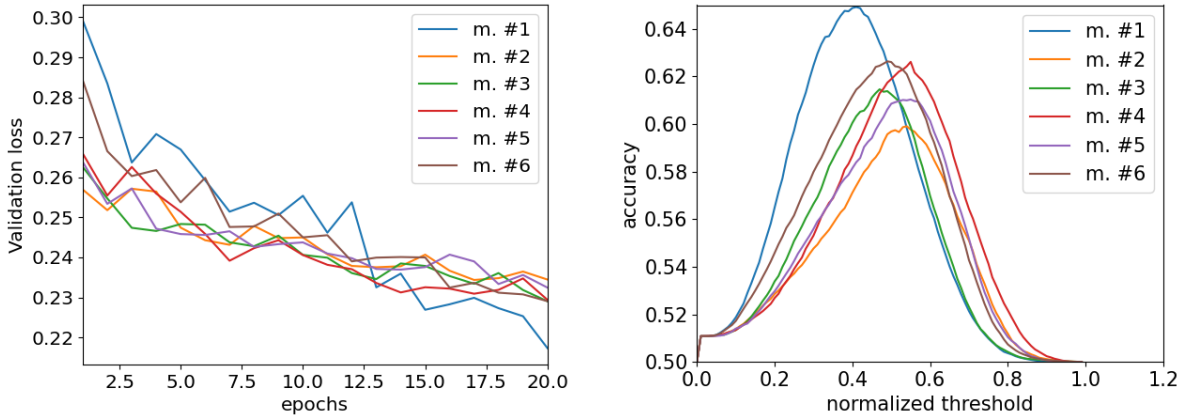


Figure 7: Comparing models with different parameters configurations, see Table 1.

The experiment showed that "model #1" configuration (see Table 1) achieved highest validation accuracy, 0.649, and the lowest loss, 0.217, after 20 epochs. Moreover, the loss decay curve is the steepest for this model thus suggesting high expected gains in the future epochs. Perhaps, use of 1024 neurons in fully connected layers in other models, has

	model 1	model 2	model 3	model 4	model 5	model 6
1^{st} fully connected	2048	1024	1024	2048	1024	1024
2^{nd} fully connected	2048	1024	1024	1024	1024	2048
Embedding length	96	128	96	96	128	96
Feature depth block #1	32	64	32	32	64	32
Feature depth block #2	79	104	84	109	124	76
Feature depth block #3	173	127	241	195	166	107
Feature depth block #4	422	171	440	410	458	418
Feature depth block #5	449	238	454	426	474	475
Final validation loss	**0.217**	0.234	0.229	0.229	0.232	0.229
Best validation accuracy	**0.649**	0.599	0.615	0.623	0.610	0.626

Table 1: Summary of six VGG-16 configurations tested during random search.

constrained their capacity by too much. Another observation to make is that both models with 64 channels in the first convolution block, achieved lowest accuracy. Perhaps, due to lower resolution of the image, there are not as many features to extract and thus 32 feature map layers are sufficient. Since images present same semantic meaning as their 256x256 original copies, we still need to use high depth for the later convolutional blocks, to recover these meanings.

One could correctly argue, that our random search has not covered parameter search space thoroughly, however due to time constraints of this project and quite high accuracy achieved for model 1, we decide to accept this potentially sub-optimal configuration.

3.2 ResNet

Deeper CNN architectures have better representative power, due to effective hierarchical learning, where every next layer forms features which are a function of local features from previous layers. Decreasing network width and increasing depth, allows networks to achieve equally (or better) performance while lowering the computational cost and number of parameters. However, if we tried to make VGG deeper its performance would suffer from vanishing gradients. A product of many small partial derivatives ($<< 1$) goes to zero as we include more factors in our chain rule, which grows in length with number of layers. Therefore updates of weights in early layers would be really ineffective, preventing network from learning. **ResNet** architecture [12] has solved this problem, by introducing skip connections. It effectively computes $y = f(x) + x$ rather then $y = f(x)$ which improved flow of gradient to earlier layers.

We have implemented two variants of ResNet network, ResNet-50 and ResNet-18. For both networks, we have reduced the stride of the first convolutional layer to 1 and removed the first max pooling layer, to maintain more spatial size of the image for later layers. Due to the size of ResNet-50 we were restricted to batch-size of 32, as higher batches didn't fit into GPUs memory. We have found that choosing a right learning rate for both networks was very important as too high learning rate either made the loss diverge in early steps or lead to a poor local optima early on in the training. We used 0.001 for ResNet-50 and 0.0001 for ResNet-18.

Figure 8 shows that best VGG-16 configuration has outperformed both ResNet models. This is despite the lower time it took to train, see Table 2. Training time performance can be explained by looking at batch size and the number of flops. Our ResNet-50 does 9.8b

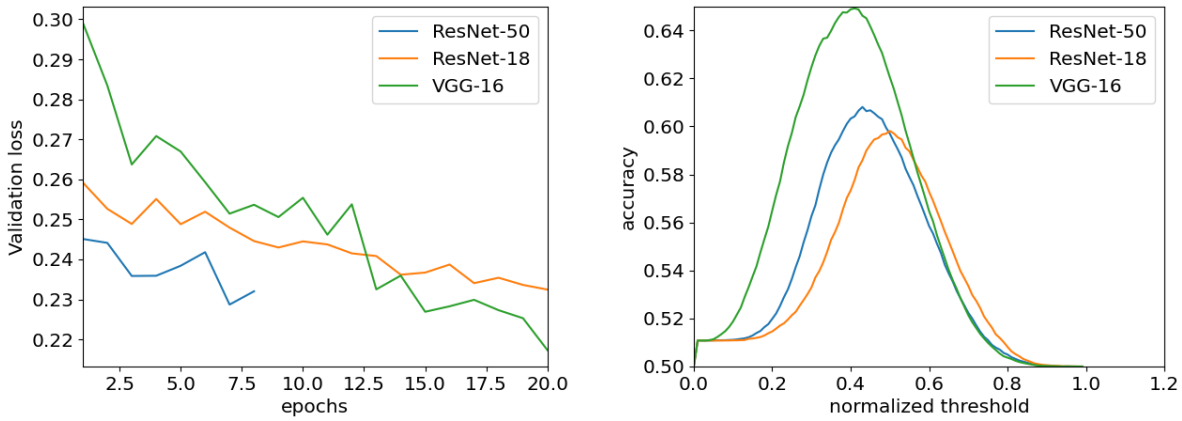


Figure 8: Comparison of validation loss and accuracy for ResNet and VGG-16 networks.

flops, 13 times more then ResNet-18 and 7.6 times more then VGG-16. Additonally, VGG used mini-batches of 64 which reduced number of steps per epoch. It could also benefit from higher learning rate 0.001, which could be used, as we didn't observe loss explosion or convergence to a local optima in early phase of training. It is surprising to see ResNet-50 perform as many as 9.8b flops, since the original paper states that, on 256x256 pictures, it does only around 4b flops. This could be due to some error in our implementation which we couldn't trace down, and due to time constraints of the project and promising results for VGG, decided to stop pursuing.

We are satisfied with VGG-16 performance in this experimentation and will use it in the final model.

	ResNet-50	ResNet-18	VGG-16
Epochs trained	8	20	20
Time to train	4.5h	1.5h	1h
Initial learning rate	0.001	0.0001	0.001
Num. parameters	23.8 mln	6.4 mln	29.0 mln
batch size	32	32	64
FLOPS	$9.88 * 10^9$	$0.76 * 10^9$	$1.29 * 10^9$

Table 2: Training and model data for ResNet models and custom VGG-16 architecture.

4 Training procedure

Data preprocessing All image pixel values were scaled down to $[0, 1]$ range, to make the learning more stable. Default Glorot (Xavier) weight initialisation, which we use, is independent of input magnitude, therefore a large 255 pixel value could result in excessively large activation and thus direct the network in a wrong direction. As suggested by VGG-16 paper [11], we also zero center our data by subtracting mean pixel values from each pixel, to speed up the training convergence.

Hyper-parameters For our final model, contrastive loss with VGG-16 backbone, we used mini-batch size of 64 and a 0.001 learning rate for first 20 epochs, and 0.0005 for all epochs afterwards (we use default epoch size, each epoch iterates over all images).

Data augmentation Data augmentation, such as: cropping, rotation, scaling, translation, horizontal flipping, or contrast, illumination, and gamma adjustments, can artificially enlarge the dataset and, thus, improve final model performance and reduce overfitting.

However, we have decided not to use data augmentation in this project. With N^2 possible image pair combinations, which is 2 Billion for 45,000 image dataset, we believe there to be enough data variability to achieve good performance and we haven't observed need for extra regularisation.

Naive pair matching Initially, we formed image pairs randomly, offline, maintaining 50-50 balance between similar and dissimilar pairs. These would be then loaded for training, all at once.

Such approach has two problems. First, it is time consuming to form and load large datasets and its size has to be predefined. We solve this by implementing **Data Generator**. Second, as the learning progresses the network learns image embeddings and therefore more and more image pairs become trivial and stop contributing to the loss function slowing down convergence, this problem can be alleviated via **Pair mining**.

Data Generator Training with Data Generator allows us to pair images while loading mini-batch for training, rather than having to prepare pairs prior to model fitting. With such approach we don't have to specify image pairs in advance and can potentially explore all $2 * 10^9$ possible pairs. Therefore this approach not only speeds up experimentation time, but also allows us to potentially explore all possible pairs, which we have observed, was important in avoiding overfitting.

Pair Mining Slow convergence, caused by trivial data pairs, can be improved by being smart about choosing pairs for training. For pairs of similar images we want to select such pairs for which the distance is large (high intra-class variance), as these will maximise pair contribution to model update. Such pairs are called hard-positives. For negative pairs, we want to consider pairs of image classes which are clustered closely together (low inter-class variance) as the model will benefit most from separating overlapping clusters, rather than separating clusters which are already far away. These are hard-negative pairs.

Pair mining is an extensive area of research in Deep Metric Learning with multiple algorithms available [9], [10]. One can choose between online mining, where best pair matches are found within a mini-batch, and offline mining, where best pairs are selected from the entire dataset based on most recent model checkpoint.

In this project we explore the later option. We focus only on mining hard-negative pairs, which is computationally less expensive (when number of classes << number of samples per class) and as we have observed in Section 2, contrastive loss performs well on learning intra-class variances. Our approach is summarised in the psudo-code below.

Algorithm 1 Custom Hard-Negative pair mining

procedure UPDATING RELATIVE DISTANCES BETWEEN CLUSTERS
    load and store all training images (45,000)
    start pairing images randomly
    while training, every epoch **do**
        if *epoch count* > ***K*** **then**
            1. Load most recent model checkpoint
            2. Get feature extractor layer from the model
            3. Predict embedding vectors for train data
            4. Compute mean cluster embedding for each class (100 classes)
            5. Compute 100x100 distance matrix between all class centers
            6. For each class store a list of **M** closest class clusters
            7. set **K** to 0
        K += 1

procedure PAIR SAMPLING
    for image *in* mini-batch **do**
        set *pair* randomly to 0 or 1
        if *pair* == 1 (a positive pair) **then**
            1. From training samples of the same class pick one at random
            (resample if the same image as *image*)
        if *pair* == 0 (a negative pair) **then**
            1. Pick class, at random, from a list of **M** closest classes, to *image* class.
            2. From samples for that class pick one at random

Unfortunately, the algorithm we have devised did not produce satisfactory results. We hypothesize that the poor performance was most likely caused by restricting negative pairs to be generated only with M closest categories for K next epochs. This has perhaps intensified the contrastive loss limitation mentioned in Section 2, where by separating given category from M closest categories we might be unwillingly moving it closer to other categories, which were far previously, but are not being considered for K epochs.

To verify this, we ran the algorithm for different M values, as illustrated in Figure 9. We start from a model checkpoint with 79.9% accuracy, to ensure we form viable category clusterings. As expected, for low M values the accuracy drops, as we are potentially decreasing separation between $M' = (100 - M)$ remaining categories. Only for M as high as 80 the accuracy seems to be somewhat improving with epochs.

This issue could be solved by either computing some less expensive category clustering at more frequent rate, or by considering online pair mining where for each mini-batch we sample P images from N classes, and form hard-negative pairs with closest class out of N classes, only for a given mini-batch. Unfortunately we ran out of time to implement these improvements.

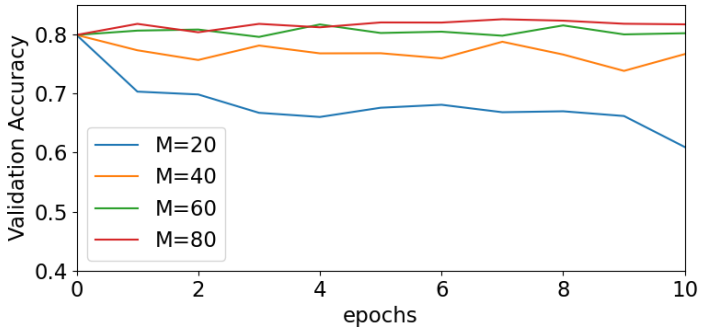


Figure 9: Accuracy evolution over 10 epochs, after a model with 79.9% accuracy was trained with our custom Hard-Negative pair mining algorithm. M closest clusters for each category were updated before 1^{st} and 5^{th} epoch.

Network regularisation First we decided to train our network with no regularisation to check the capacity of our VGG-16 feature extractor and observe at which point it starts to overfit. We would then apply L2 weight decay and dropout as needed. However, the network did not show any signs of overfitting, Figure 10 shows that the generalisation gap grows with numbers of epochs, yet model is under-fitting. This could be attributed to high N^2 number of potential pairs which Data Generator can form. After 320 epochs, model has seen $45,000 * 320 = 14,400,000$ image pairs, which comprises only 0.71% of all possible pairs. Therefore there is a high chance the model saw majority of pairs only once.

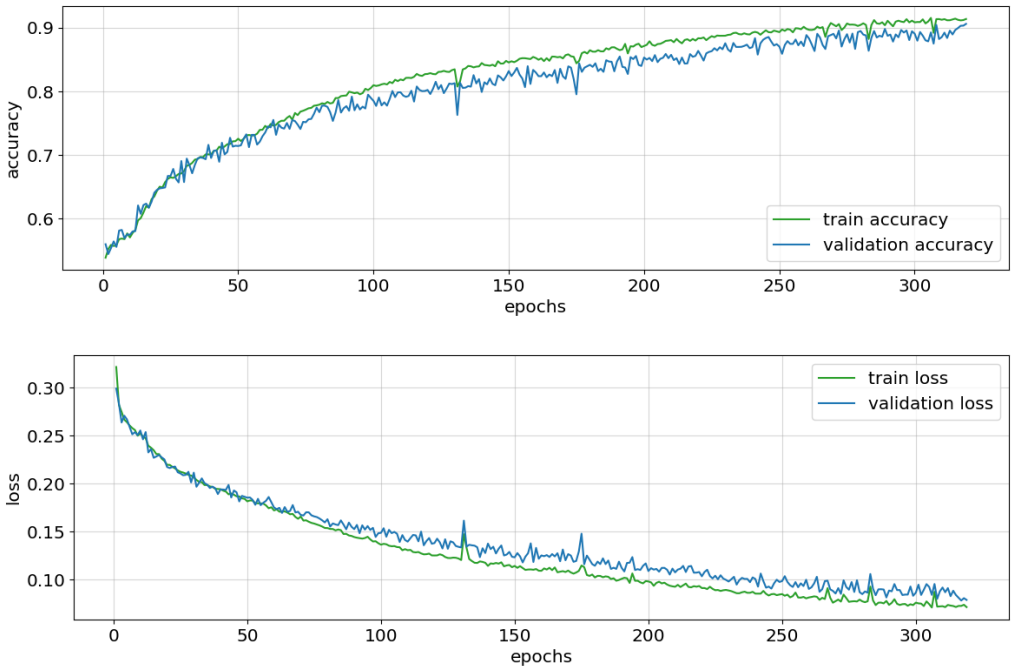


Figure 10: Evolution of validation loss and accuracy with training epochs.

5 Final model evaluation

For our final model we used contrastive objective function with VGG-16 for feature extractor (with parameter identified in Section 3). Our model achieves 90.7% accuracy on test set with 100 seen categories, and 63.8% on 100 unseen categories, see Figure 11.a (4 positive and 4 negative pairs were formed, at random, for each of 5000 test images). Figure 10 shows that after 320 epochs the validation learning slope was still positive, suggesting a better performance was possible had we continued training.

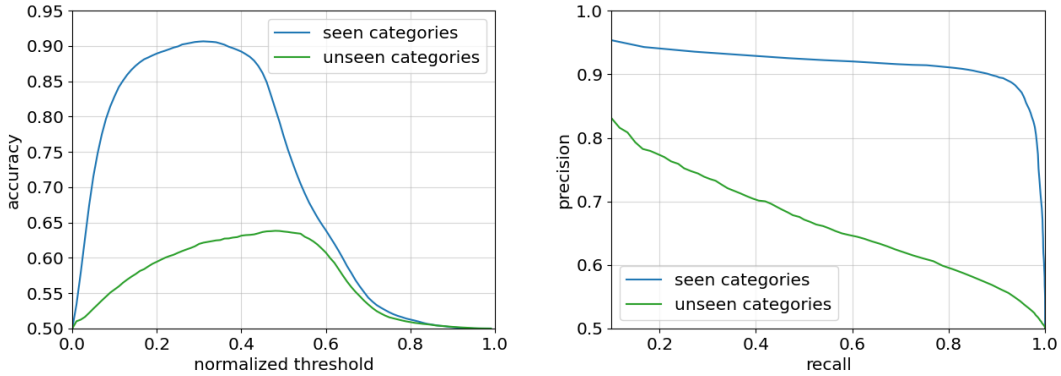
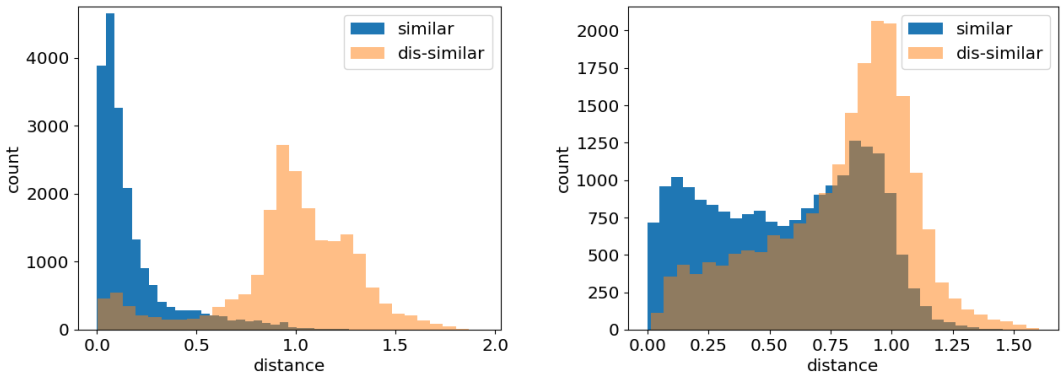


Figure 11: Test accuracy and precision-recall curve for unseen and seen categories.

We decided to use a global distance threshold. A category specific threshold would require knowledge of category of a query image at test time, while using a clustering method, would constrain the number of categories we could query to those in the validation set.

We choose accuracy as our primary metric, as, for a balanced dataset, maximising accuracy is equivalent to maximising model correctness, or alternatively reducing the sum of false accept and false reject rates. However it is important to note that selection of distance threshold can be arbitrary. Depending on the application of the image matching system, one may prioritise Low False accept rate instead, i.e. in security systems. In such case, one could choose distance threshold which guarantees higher precision, Figure 11.b, by sacrificing on recall.



(a) seen categories (b) unseen categories

Figure 12: Distribution of distances for test pairs, grouped by similar and dis-similar pairs.

Figure 12 shows distribution of distances between similar and dis-similar test pairs.

Average distance between distributions for similar and dis-similar pairs was 0.78 for seen categories and 0.22 for unseen categories. We have noted that for earlier checkpoints the model had slightly better performance on unseen data (65% accuracy). As we continued training, the model was becoming more (falsly) confident that some of the positive pairs (for unseen categories) are dis-similar, indicated by the second peak of blue distribution, Figure 12.b. This could be caused by insufficient number of general features learned by the feature extractor, making it unable to recognise pairs from the second blue peak as similar. We could have improved model performance by including more categories during training, it would be also useful to track how validation accuracy changes for unseen categories during training.

5.1 Detailed look at model performance

In this section we evaluate category specific accuracy for each of the 100 seen and 100 unseen categories. Figure 13 shows distribution of accuracies across categories, evaluated on 1000 image pairs per category - 10 positive and 10 negative pairs were selected at random, for each test image.

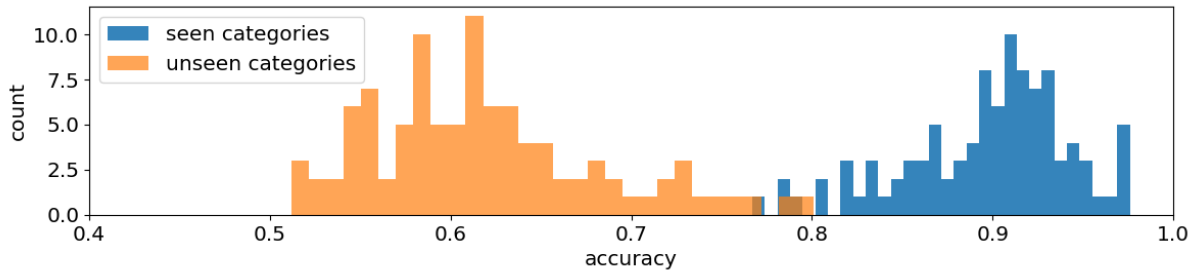


Figure 13: Distribution of accuracies for 100 seen and 100 unseen categories.

	SEEN				UNSEEN			
Rank	Category	Acc.	FAR	FRR	Category	Acc.	FAR	FRR
1	bullet train	97.6%	0%	4.8%	viaduct	79.4%	14.8%	25.6%
2	pizza	97.2%	0%	5.6%	lion	77.5%	20.2%	22.4%
3	lemon	97.2%	0%	5.6%	bison	76.4%	14.6%	31.4%
...	...	...	...	...	...	...		
98	lampshade	78.6%	15.0%	27.8%	cliff dwelling	51.3%	17.8%	79.6%
99	soda bottle	78.4%	18.6%	24.6%	chain	50.6%	23.2%	75.6%
100	backpack	76.7%	17.0%	29.6%	binoculars	50.5%	20.2%	78.8%

Table 3: Accuracies, False Accept rate (FAR), and False reject rate (FRR) for 3 best and 3 worst performing categories, both seen and unseen.

Table 3 shows that bullet train and pizza are amongst the highest scoring classes. Looking at Figure 14 we could hypothesise that good classification performance on these images, is due low intra and high inter category variance. Both bullet train and pizza have distinctive features which are present on most images (low $FRR \approx 5\%$) and are unique to that category (low $FAR = 0\%$). On the other hand, images of backpack or binoculars have diverse background and little in common (high FRR indicates high intra-class variance), and could be easily labeled as other categories, i.e. human (high FAR could be explained

by similarity to other classes). High scores for lion and bison (unseen categories), could be attributed to high presence of animals amongst seen categories. Feature common with seen animals, as well as, low intra-class variance, yielded relatively low FRR for lion and bison. For binoculars, high intra-class variance and little common features with seen categories had resulted in high FRR for that category.

(a) bullet train

(b) pizza

(c) backpack

(d) lion

(e) bison

(f) binoculars

Figure 14: For each of listed categories, 6 images were drawn at random.

References

[1] E. P. Xing, M. I. Jordan, S. J. Russell, A. Y. Ng, Distance metric learning with application to clustering with side-information, in: Advances in neural information processing systems, 2003, pp. 521–528

[2] Chopra, S., Hadsell, R., LeCun, Y., Learning a similarity metric discriminatively, with application to face verification In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005.

[3] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In Advances in neural information processing systems, pages 1473–1480, 2006.

[4] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In S. Thrun, L. Saul, and B. Schölkopf, editors, NIPS, pages 41–48. MIT Press, 2004. 2

[5] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In Advances in Neural Information Processing Systems, pages 1857–1865, 2016

[6] Picon, A. and Medela, A. (2020). Constellation loss: Improving the efficiency of deep metric learning loss functions for the optimal embedding of histopathological images. Journal of Pathology Informatics, 11(1), p.38.

[7] Schroff, F., Kalenichenko, D., Philbin, J. FaceNet: A unified embedding for face recognition and clustering In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[8] Wang, Jian Zhou, Feng Wen, Shilei Liu, Xiao Lin, Yuanqing. (2017). Deep Metric Learning with Angular Loss. 2612-2620. 10.1109/ICCV.2017.283.

[9] Oh Song, H.; Xiang, Y.; Jegelka, S.; and Savarese, S. 2016. Deep metric learning via lifted structured feature embedding. In CVPR.

[10] Wang, X., Hua, Y., Kodirov, E., Hu, G. and Robertson, N.M. (2019). Deep Metric Learning by Online Soft Mining and Class-Aware Attention. Proceedings of the AAAI Conference on Artificial Intelligence, 33, pp.5361–5368.

[11] Simonyan, Karen Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.

[12] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.