CV Project: Deep Variational Metric Learning

Team: *Goal Diggers*

Abhishek Prusty (**20161112**)
Kunal Garg (**20161067**)
Animesh Sahu (**20161028**)

Github Link: Deep Variational Metric Learning

Contents

Introduction

A metric is a function that defines a distance between each pair of elements of a set. Metric learning learns a metric function from training data to calculate the similarity or distance between samples. From the perspective of feature learning, metric learning essentially learns a new feature space by feature transformation. Given a dataset $X = [x_1, x_2, x_3, ...x_n]$, metric learning aims to find a low-dimensional subspace W to map each x_i to y_i , where $y_i = Wx_i$, such that some characteristics are preserved. In deep metric learning we train a deep neural networks to produce discriminative embedding features by taking advantage of the nonlinear feature representation learning ability of deep learning networks. The aim is find a good metric produces embeddings where samples from the same class have small distances and samples from different classes have large distances.

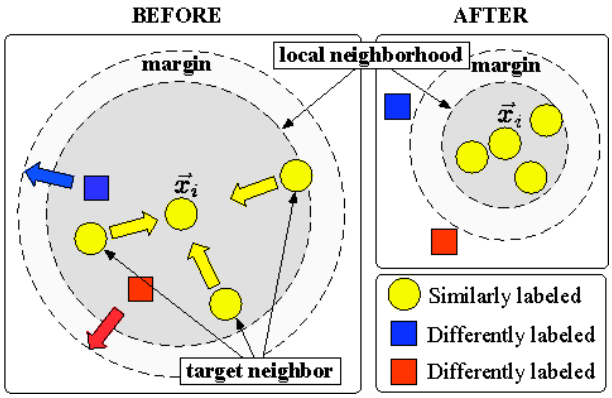

Figure 1: Metric Learning

Motivation

In the conventional metric learning methods, intra-class variance and class centers are entangled, which brings two limitations:

- Given a limited dataset with a large range of variance within classes, current metric learning methods are easily overfitting and lose discriminative power on unseen classes.
- Without disentangling intra-class variance and class centers, current methods learn a metric by exploring the boundary among classes, which means numerous easy negative samples contribute little to the training procedure.

There are intra-class variances, such as pose, view point, illumination, etc, and a robust model should be able to handle these variances. However given a limited training set, deep models will easily be overfitting if we force it to be indiscriminating to these intra-class variances. For example, in image classiication, if the illumination of the object region varies too much among different samples, the model would probably be trained to ignore these important parts but to classify the training samples from their backgrounds.

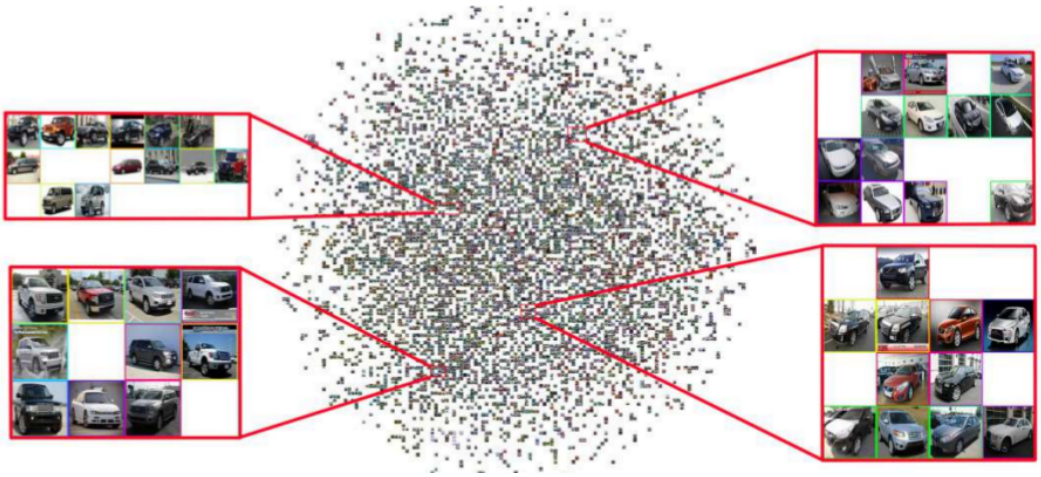

Figure 2: A similar pose change in samples from different classes leads to a cluster in the central latent space.

Proposed Approach

To overcome these problems, paper proposes a Deep Variational Metric Learning (DVML) framework to explicitly model the intra-class variance and disentangle the intra-class invariance. The paper assumes that the distribution of intra-class variance is actually independent on classes and tries to model intra-class variance with an isotropic multivariate Gaussian.

The Embedding features of samples from the same class consist of two parts; One represents the intra-class invariance, and the other represents the intra-class variance which obeys the identical distribution among different classes.

Using the distribution of intra-class variance, we generate potential hard samples from easy samples by adding intra-class variance to it which makes the model robust to pose, view point, illumination, etc.

Preliminaries

Here we describe the triplet loss which has been used as metric learning loss function in our implementation. The goal of the triplet loss is to make sure that:

- Two examples with the same label have their embeddings close together in the embedding space
- Two examples with different labels have their embeddings far away.

However, we don't want to push the train embeddings of each label to collapse into very small clusters. The only requirement is that given two positive examples of the same class and one negative example, the negative should be farther away than the positive by some margin. To formalise this requirement, the loss will be defined over triplets of embeddings:

- **Easy triplets:** triplets which have a loss of 0, because $d(a,p) + margin < d(a,n)$
- **Hard triplets:** triplets where the negative is closer to the anchor than the positive, i.e. $d(a,n) < d(a,p)$
- **Semi-hard triplets:** triplets where the negative is not closer to the anchor than the positive, but which still have positive loss: $d(a,p) < d(a,n) < d(a,p) + margin$

Triplet mining: The idea of online triplet mining is to compute useful triplets on the fly, for each batch of inputs. Given a batch of B examples (for instance B images of cars), we compute the B embeddings and we then can find a maximum of B^3 triplets. Of course, most of these triplets are not valid (i.e. they don't have 2 positives and 1 negative).Whenever we have three indices $i, j, k \in [1, B]$, if some examples i and j have the same label but are distinct, and example k has a different label, we say that (i,j,k) is a valid triplet. In out case, we mine only semi-hard triplets(see 2.1).

The final Triplet loss [3] is,

$$L_m = max(d(a,p) - d(a,n) + margin, 0)$$

DVML framework

Feature Extractor

We use GoogLeNet [4] as the backbone for feature extractor. This feature extractor is fed an input of pre-processed data. Then we add three parallel

fully-connected layers after the average pooling layer of GoogLeNet, with output dimension of 128 (in paper they use 512) due to memory issues. We demonstrate that this embedding size can give comparable results as the paper .Out of the three FC networks, two of them are used to approximate μ and $log\sigma^2$. The third fully connected layer is used to learn intra-class invariant features, namely, the class centers, which is also the output of the model.

Generator

The Generator takes as inputs the class centers(z_I) and the features sampled(z_V) from the learned distribution $\mathcal{N}(z_V; \mu^{(i)}, \sigma^{2(i)}I)$ and then outputs element-wise sum of them as synthesized discriminative samples.

Decoder

Decoder is a fully connected network with two layers with output dimension 512 and 1024 respectively, which takes as inputs the synthesized features(128 dimension) generated by the generator. The 1024 dimensional vector that this network outputs is used to reconstruct the output features of the CNN's last average pooling layer(1024 dimensional).

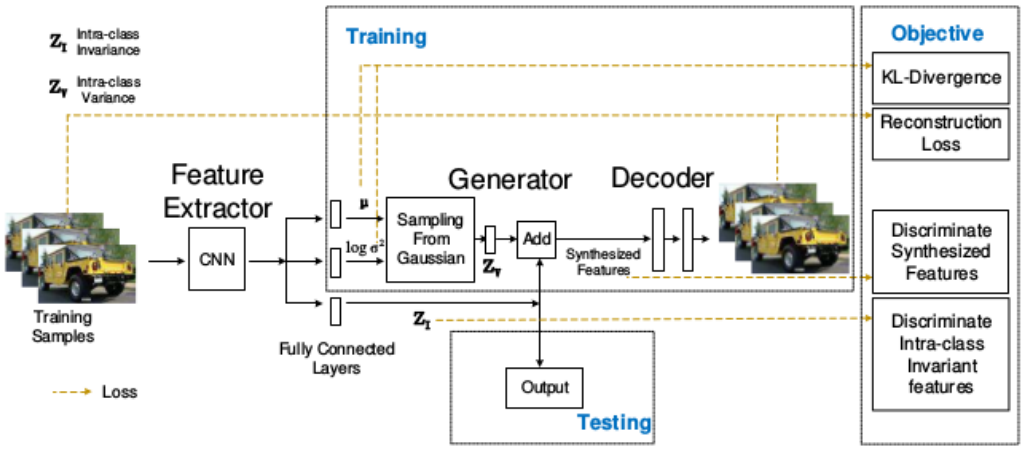

Figure 3: DVML Model Architecture

Objective Functions

The training procedure of DVML is simultaneously constrained by the following four loss functions:

- L1: KL divergence between learned distribution and isotropic multivariate Gaussian
- L2: Reconstruction loss of original images and images generated by the decoder
- L3: Metric learning loss of the combination of of sampled intra-class variance and learned intra-class invariance.
- L4: Metric learning loss of learned intra-class invariance

The first two losses ensure that the intra-class variance shares the same distribution and does contain sample-specific information of each image sample. The third ensures a robust boundary among classes and the fourth ensures that the intra-class invariance represents a good class center for each class.

KL Divergence Loss

Kullback–Leibler divergence (also called relative entropy) is a measure of how one probability distribution is different from a second. For discrete probability distributions P and Q defined on the same probability space, the KL divergence between P and Q is defined to be

$$D_{KL}(P||Q) = \sum_{x \varepsilon X} P(x) log(P(x)/Q(x))$$

Here we let the prior distribution of z_V is the centered isotropic multivariate Gaussian, $P(z_V) = \mathcal{N}(z_V; 0, I)$. For the approximation posterior, we let it be a multivariate Gaussian with a diagonal covariance, $Q(z_V|x^i) = \mathcal{N}(z_V; \mu^{(i)}, \sigma^{2(i)}I)$ We use the outputs of fully-connected layers, to approximate the mean and s.d. of the posterior, $\mu^{(i)}, \sigma^{(i)}$.

We sample z_V from Guassian and we want to train VAE with gradient descent but the sampling operation doesn't have gradient!. So, to make the network differentiable we use reparameterization trick i.e. KL-divergence between a diagonal multivariate normal(Q), and a standard normal distribution(P) can be estimated from μ and σ by this function:

$$D_{KL}(\mathcal{N}((\mu_1, \mu_2, ..., \mu_k)^T, diag(\sigma_1^2, \sigma_2^2, ..., \sigma_k^2))||\mathcal{N}(0, I)) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_i^2 + \mu_i^2 - log(\sigma_i^2) - 1)$$

$$L_1 = \frac{1}{2B}\sum_{i=1}^{B}\sum_{j=1}^{J}((\sigma_j^{(i)})^2 + (\mu_j^{(i)})^2 - log((\sigma_j^{(i)})^2) - 1)$$

L_1 enforces the distribution of intra-class variance to be isotropic centered Gaussian.

Reconstruction Loss

Reconstruction loss is the reconstruction error of an image x when reconstructed from synthesized features. Due to the high cost of image reconstruction, we use the output features of GoogLeNet's last average pooling layer as the reconstruction target and calculate the L2 loss between synthesized featured and target for each batch (B represents Batch Size).

$$L_2 = \frac{1}{B}\sum_{i=1}^{B}||x^{(i)} - \hat{x}^{(i)}||_2$$

L_2 ensures the intra-class variance preserve sample- specific information.

Discriminate Synthesized Features

Potential hard samples from easy negative samples can be generated with the learned distribution of intra-class variance by adding the embedding features of easy samples with an biased term sampled from the distribution of intra-class variance. We take synthesized embedding features which are the output of the generator and take them as the inputs of metric learning loss function. In our implementation we used Triplet loss function but we can also use N-pair loss to evaluate metric learning loss.Thus our third loss becomes

$$L_3 = \mathcal{L}(z_I + z_v)$$

Discriminate Intra-class Invariant features

z_I are the intra-class invariance features(given by the third layer of the Feature Extractor) and also the output of our model in testing. This loss enforces the intra-class invariance, namely the class centers, to be discriminative and is calculated by tripled loss function which takes z_I as input.

$$L_4 = \mathcal{L}(z_I)$$

The final objective function is:

$$L = \lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3 + \lambda_4 L_4$$

Dataset : Cars196

The Cars dataset [1] contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split.

Figure 4: Cars196 sample images

Implementation details

We use tensorflow as our framework of choice to implement this paper. Refer to figure below for the computational graph of the implementation visualized using tensorboard.

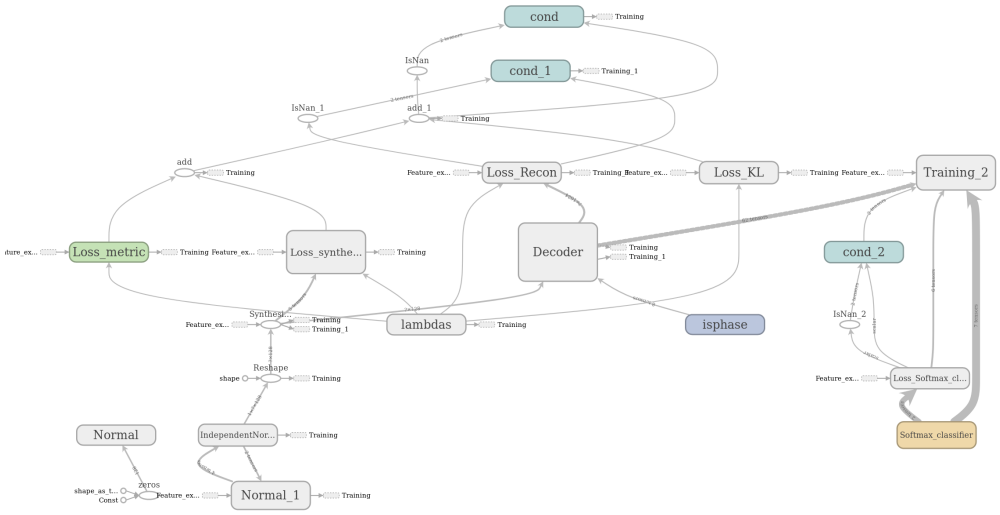

Figure 5: Computational Graph

Data Pre-processing

We use hdf5 format to store the images along with their class labels. For pre-processing, the images are normalized into 227 x 227 and then random crop and horizontal mirroring is performed for data augmentation as mentioned in the paper. We use a 50:50 train-test split(labels 1 98 for train, 99 196 for test).

Training

Training procedure of DVML is simultaneously constrained by the following four loss functions described in 2.3. There are two phases in the training procedure. In the first phase, we cut of the back-propagation of the gradients from the decoder network for the stability of the embedding part(by setting training parameter of tensorflow layers as False). In the second phase, we release the constraint. We have taken embedding size as 128 and also the batch size as 128. We take Adam as the optimizer and set the training rate for Feature extractor part to be $7e-5$, for decoder to be $1e-2$ and for cross entropy loss to be $1e-3$. We trained the model using various different values of λ_1,λ_2,λ_3 and λ_4.

Evaluation

For quantitative evaluation, three metrics, namely ,Recall@K, NMI(Normalized Mutual Information) and F-score. These metrics are computed for the embedding space which the network outputs.

Normalized Mutual Information

NMI is good measure for determining the quality of clustering,It is basically an external measure because we need the class labels of the instances to determine the NMI.Since it's normalized we can measure and compare the NMI between different clusterings having different number of clusters, even If we have to compare two clustering that have different number of clusters we can still use NMI.

$$NMI(Y, C) = \frac{2 \times I(Y; C)}{[H(Y) + H(C)]}$$

where, Y = class labels, C = cluster labels, H(.) = Entropy, I(Y;C) = Mutual information between Y and C.

Recall@K

Recall calculates how many of the Actual Positives the model captures through labeling it as Positive (True Positive). In our case, Each test point in the embedding space first retrieves K nearest neighbors from the test set and receives score 1 if an point of the same class is retrieved among the K nearest neighbours and 0 otherwise.A true positive (TP) decision assigns two similar documents to the same cluster, a true negative (TN) decision assigns two dissimilar documents to different clusters. There are two types of errors we can commit. A (FP) decision assigns two dissimilar documents to the same cluster. A (FN) decision assigns two similar documents to different clusters.

$$Recall@K = \frac{TP}{TP + FN}$$

F1 Metric

F1 score is the harmonic mean of Precision and recall.

$$F1 - score = \frac{2 \times Precision.Recall}{Precision + Recall}$$

Results

As we mentioned in the section 4.2, we have used two phases for training. However, the values for λ_is have to be empirically determined through experiments. In this section, we report the results obtained in each of the experiments using loss curves obtained using tensorboard. We report the final metrics that we mentioned in 4.3 towards the end of the section in table 6.

Experiment-1

For Phase1:

$$\lambda_1 = 1, \lambda_2 = 0.5, \lambda_3 = 0.5, \lambda_4 = 1$$

For Phase2:

$$\lambda_1 = 0.8, \lambda_2 = 1, \lambda_3 = 0.4, \lambda_4 = 0.8$$

1600 iterations were performed for each Phase.

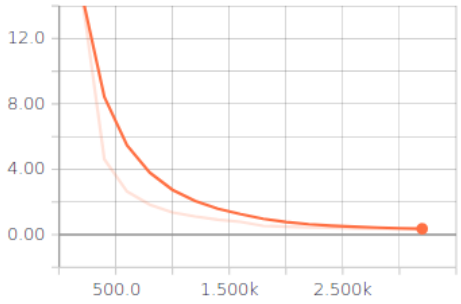

Figure 1: KL Divergence

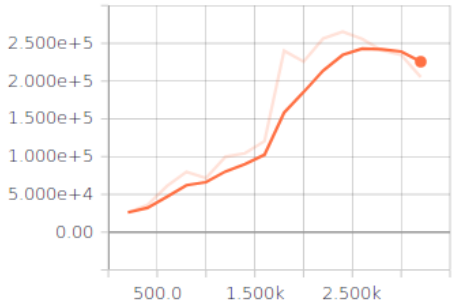

Figure 2: Reconstruction Loss

The KL divergence loss decreases rapidly in the first phase but relatively slower in the the second phase. The reconstruction loss curve indicates that the loss increases in the first phase because we are freezing the weights of the decoder in this phase. In the second phase however, the loss starts decreasing after a few iterations.

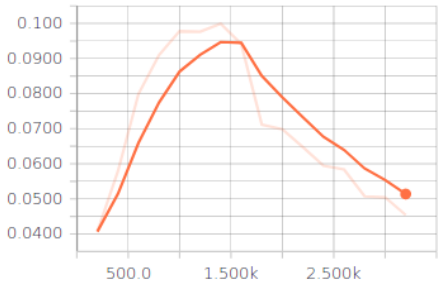

Figure 3: Discriminate Synthesized Features

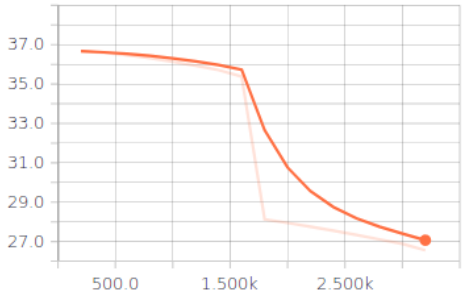

Figure 4: Discriminate Intra-class Invariant features

If we take a look at the Discriminate synthesised features loss curve, the triplet loss increases in the first phase because the above reason and decreases in a similar way in the second phase. The Discriminate Intra-class invariant features loss curve dereases slowly in the first phase and decreases quite rapidly in the second phase, indicating a good class centres prediction.

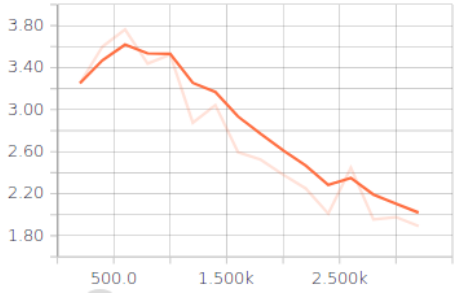

Figure 5: Cross Entropy Loss

Finally, the cross entropy loss decreases , indicating that the embedding space caters as a good input for classification.

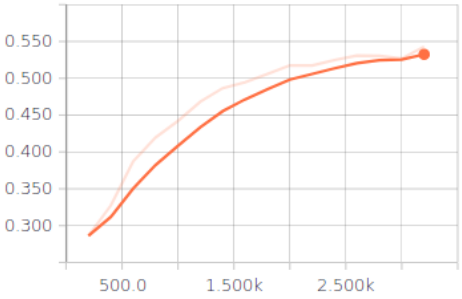

Figure 6: Recall@1

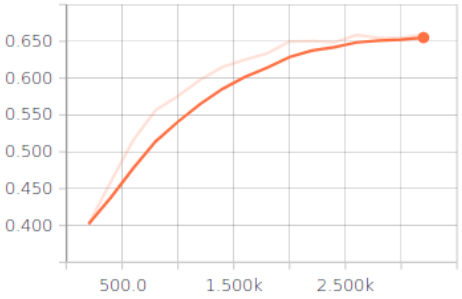

Figure 7: Recall@2

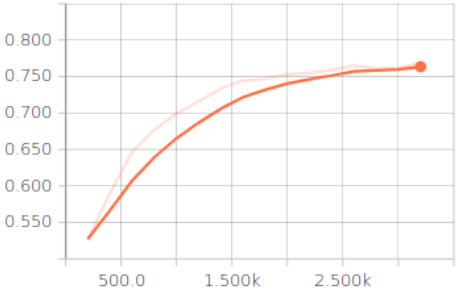

Figure 8: Recall@4

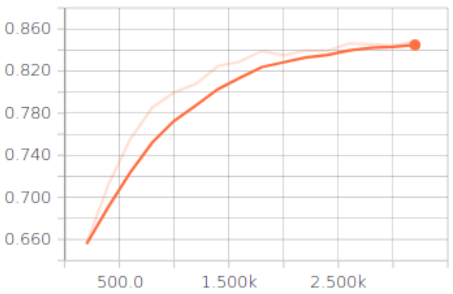

Figure 9: Recall@8

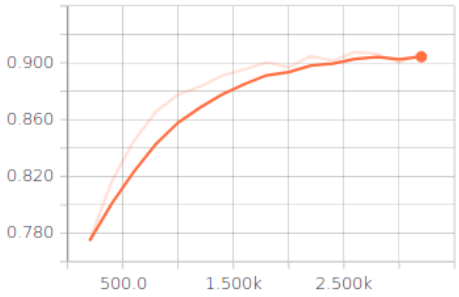

Figure 10: Recall@16

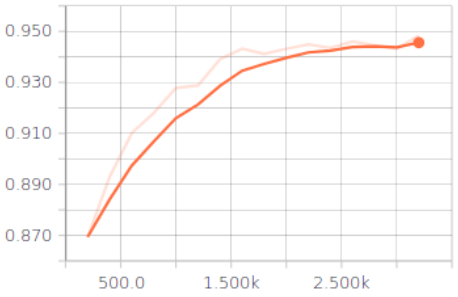

Figure 11: Recall@32

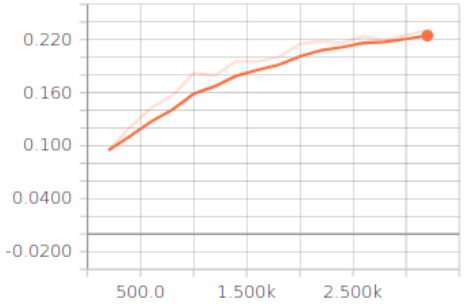

Figure 12: F1 score

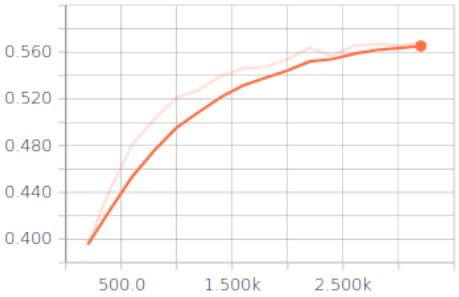

Figure 13: NMI

The recall , F1 score and NMI curves indicate a steady rise in values with iterations, which is a indicator of the fact that our embeddings produce a good clustering. The final NMI and F1 score are 0.568 and 0.2306 respectively.

Experiment-2

For Phase1:

$$\lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 0.1, \lambda_4 = 1$$

For Phase2:

$$\lambda_1 = 0.8, \lambda_2 = 1, \lambda_3 = 0.2, \lambda_4 = 0.8$$

1600 iterations were performed for each Phase.

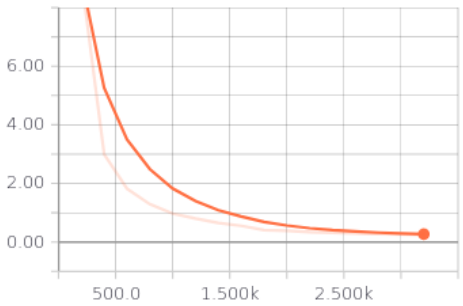

Figure 14: KL Divergence

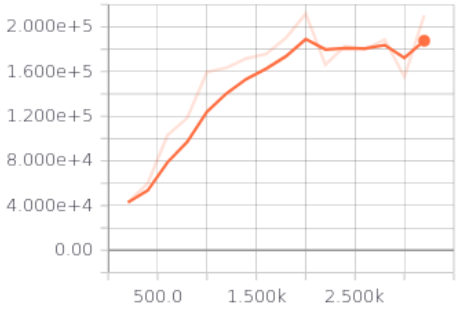

Figure 15: Reconstruction Loss

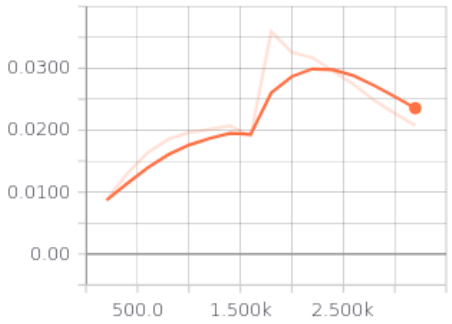

Figure 16: Discriminate Synthesized Features

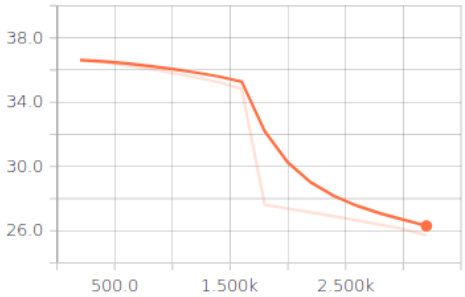

Figure 17: Discriminate Intra-class Invariant features

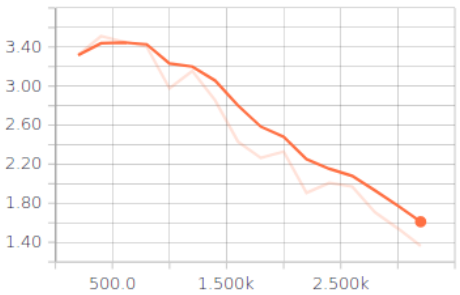

Figure 18: Cross Entropy Loss

Comparing these curves with those of experiment 1, we observe a similar trend in the loss curves, baring only the reconstruction and Discriminate synthesized features metric loss. The reconstruction loss does not decrease much in the second phase, but becomes constant after a few iterations. The metric loss of synthesized features also starts decreasing after a few iterations in the second phase of training.

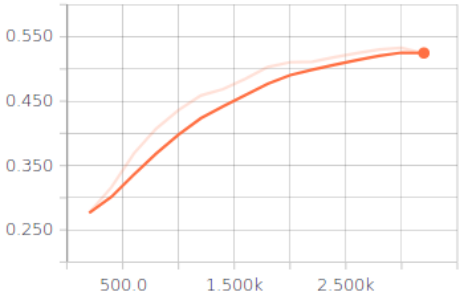

Figure 19: Recall@1

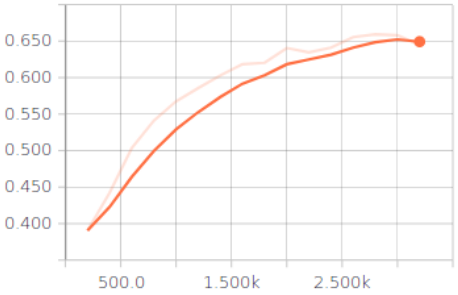

Figure 20: Recall@2

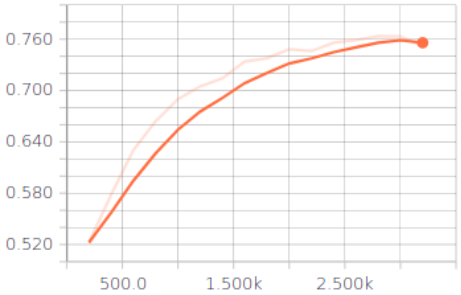

Figure 21: Recall@4

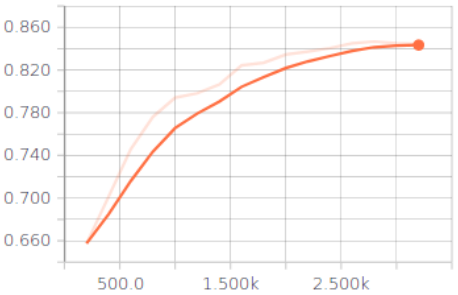

Figure 22: Recall@8

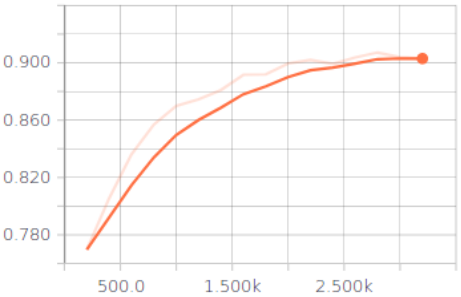

Figure 23: Recall@16

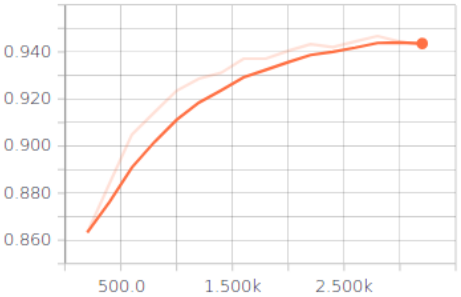

Figure 24: Recall@32

The recall curves follow the same trend as in Experiment 1.

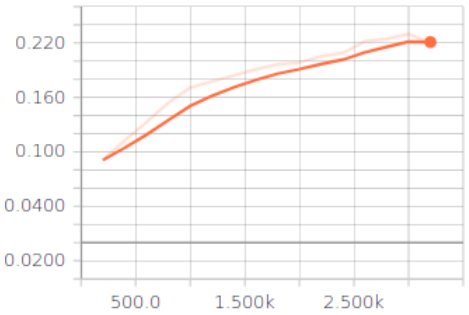

Figure 25: F1 score

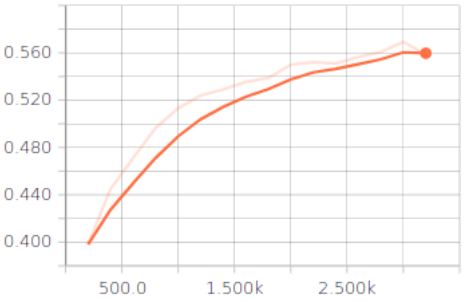

Figure 26: NMI

The final NMI and F1 score are 0.5597 and 0.2204 respectively.

Experiment-3

For this experiment, we explore the feasibility of training the whole architecture in one phase , without freezing the weights of the decoder network. We performed 3000 iterations during training. Additionally, we set all the λs to 1.

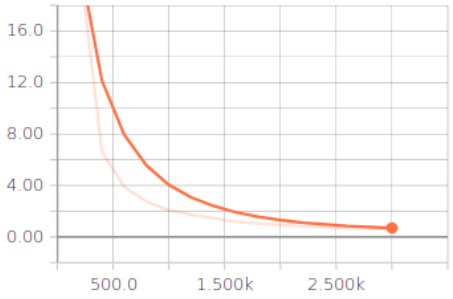

Figure 27: KL Divergence

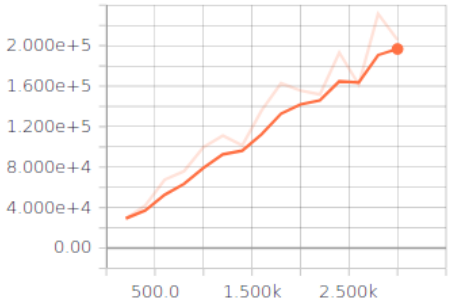

Figure 28: Reconstruction Loss

The KL Divergence loss decreases much faster than that in the previous two experiments. However, the reconstruction loss does not decrease with the number of iterations, which shows that the embedding space is not stable and hence, training has to be performed in phases as described in previous sections.

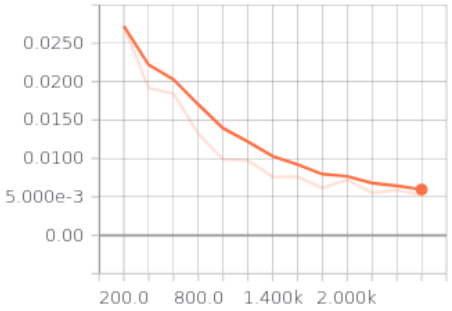

Figure 29: Discriminate Synthesized Features

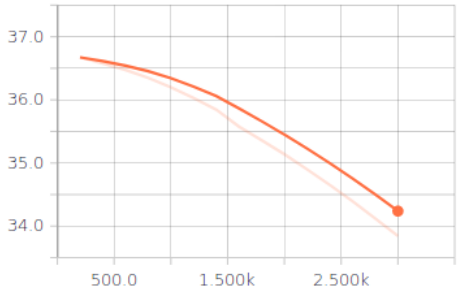

Figure 30: Discriminate Intra-class Invariant features

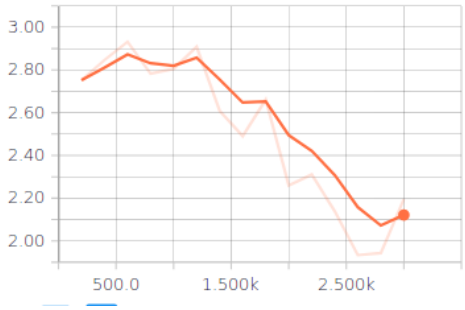

Figure 31: Cross Entropy Loss

The metric loss for Synthesized features and Intra-class invariant features decreases with the iterations. The metric loss for Intra-class invariant features decreases in almost a linear pattern as opposed to a more subtle rate in decrease in the previous experiments.

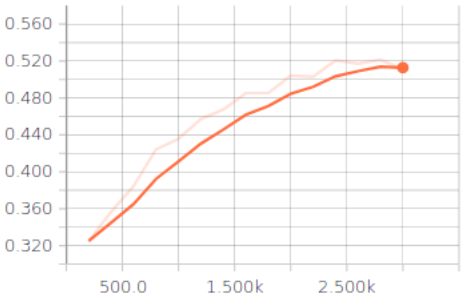

Figure 32: Recall@1

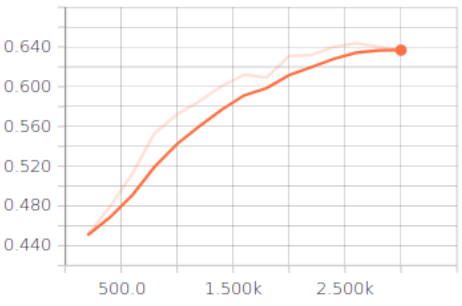

Figure 33: Recall@2

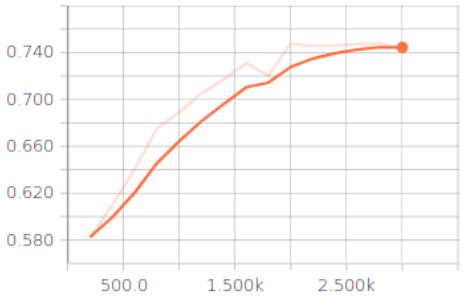

Figure 34: Recall@4

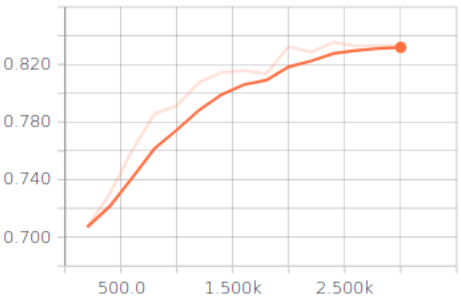

Figure 35: Recall@8

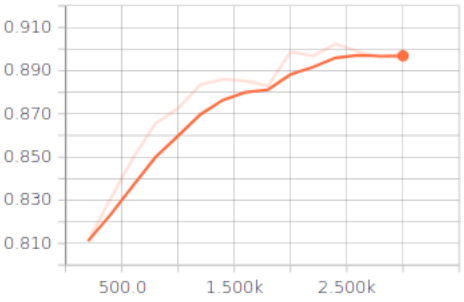

Figure 36: Recall@16

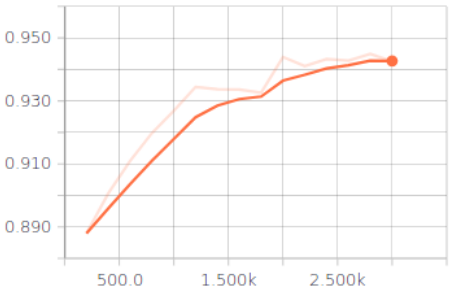

Figure 37: Recall@32

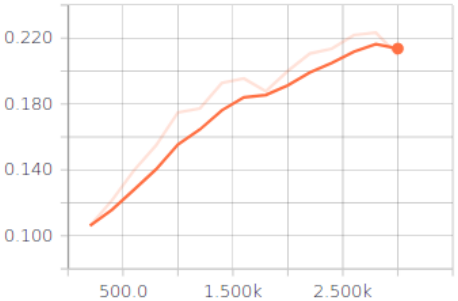

Figure 38: F1 score

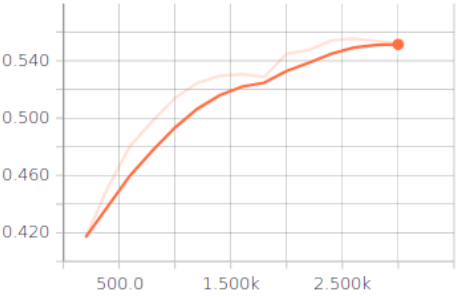

Figure 39: NMI

The recall , F1 score and NMI curves indicate that the peak which the metrics reach are not as good as the previous experiments. The final NMI and F1 scores are 0.551 and 0.2096 respectively.

Table 1: Comparison between 3 experiments

	NMI	F1-Score	R@1	R@2	R@4	R@8	R@16	R@32
Expt-1	0.568	0.231	0.543	0.659	0.769	0.847	0.906	0.948
Expt-2	0.558	0.2204	0.524	0.644	0.751	0.843	0.902	0.943
Expt-3	0.551	0.2096	0.511	0.637	0.744	0.833	0.897	0.942

Conclusions

Via this project, we explored various methods of metric learning and tried to improve upon the limitations of the conventional metric learning using techniques mentioned in the paper [2]. We report similar metrics as in the original paper and were able to achieve nearly the same values for the evaluation metrics as in the paper. Additionally, we also present the loss curves from our training experiments in order to critique the use of some of the hyperparameter choices made in the experiments preformed. Finally, we conclude by acknowledging that for the problem of metric learning, the crucial step is to pick good samples while training in order to achieve good performance, and we have successfully explored some techniques for the same.
The tensorboard checkpoints can be viewed at link

References

[1] Jonathan Krause et al. "3D Object Representations for Fine-Grained Categorization". In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.

[2] Xudong Lin et al. "Deep Variational Metric Learning". In: *The European Conference on Computer Vision (ECCV)*. 2018.

[3] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering." In: *CoRR* abs/1503.03832 (2015). URL: `http://dblp.uni-trier.de/db/journals/corr/corr1503.html#SchroffKP15`.

[4] Christian Szegedy et al. "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: `http://arxiv.org/abs/1409.4842`.