Representation and Distance Metric Learning

Eren Kopuz
CID: 01095555
ek2315@imperial.ac.uk

Rajan Patel
CID: 01062228
rnp15@imperial.ac.uk

Abstract

We present an exploration into distance metric learning, by considering the retrieval process involved in Person Re-Identification (Re-Id). Re-Id is the task of correctly identifying images of the same individual taken from disjoint camera views (i.e CCTV camera network), or from the same camera on different occasions. Our objective is to improve on baseline methods that perform K-Nearest Neighbours (K-NN) retrieval experiments, adhering to standard practices in pattern recognition [6]. We endeavour to find an optimal approach that minimizes the retrieval error from computed ranklists(i.e @Rank1,@Rank10), whilst taking into consideration computational complexity.[1]

1. Distance Metric Learning

1.1. Problem Overview

Re-Id is a popular field of research [14] due to its high practical value in a wide range of applications including surveillance [16], content-based image retrieval [13] and multi-camera event detection [10]. One of the main challenges of Re-Id originates from unwanted noise introduced in extracted features, due to intra-class variations of a given person among different camera views. Such variations may stem from variations in appearance caused by an assortment of illuminations, different poses, low resolution and partial or near-full occlusion due to crowded environments. Specifically, cluttered backgrounds make subsequent processing of an input image more difficult. There is also an interference problem of having similar samples while producing ranking results [15]. In the following sections, we present our own exploration into the Re-Id process, whilst trying to address and tackle some of the challenges stated in this section.

1.2. Dataset Characteristics

All experiments in this report have been conducted on the popular CUHK03-NP (New-protocol)[2] data-set, which contains $N = 14,096$ images representing $L = 1,467$ unique identities of pedestrians. Each unique identity is represented between 7-10 times, with images taken from two different cameras C_1 and C_2. The feature vectors of the images were obtained from the state-of-the-art Deep Convolutional Residual Neural Network: ResNet-50[2].

[1]Training and Testing instructions along with .zip file link to working code for all experiments is documented in Appendix XIII.

[2]The training instructions for this assignment state that 13,164 images of 1,360 pedestrians were given, which is true for the older CUHK03 dataset. However, CUHK03-NP was the actual data-set we received.

1.3. Preprocessing

Each feature vector for a given flattened image contains $D = 2048$ features and is represented mathematically in the form of a column-vector denoted as x_i, where $x_i \in \mathbb{R}^D$ and i $\in$ [1, 14096]. The data-set is split into a training set X_{train} and a testing test X_{test}.The cardinalities of X_{train} and X_{test} are denoted as $N_{train} = 7368$ and $N_{test} = 6728$ respectively. A subset of X_{train} is taken to form a validation set X_{val} with cardinality $N_{val} = 966$ containing $L_v = 100$ unique identities. For testing purposes, X_{test} is split further into X_{query} and $X_{gallery}$ with cardinalities of $N_{query} = 1400$ and $N_{gallery} = 5328$ respectively. Y_{train}, Y_{query} and $Y_{gallery}$ are denoted as the sets of labels (identities) for X_{train}, X_{query} and $X_{gallery}$ respectively.

1.4. Analysis of Training Features

Given that the provided features have been extracted using ResNet-50, we hypothesize that they are already likely to provide fairly accurate linear separation in high dimensional euclidean space[3]. In Figure 1, we plot the distribution of data mapped to the three dimensions of maximum variance using PCA and LDA techniques. Visually, we can conclude that the data points within the same class, appear to be grouped fairly well together, while the introduction of class labels in supervised LDA doesn't appear to provide much benefit. We can conclude that the inherent variation in the feature vectors therefore already represents the class separation in a near optimal way.

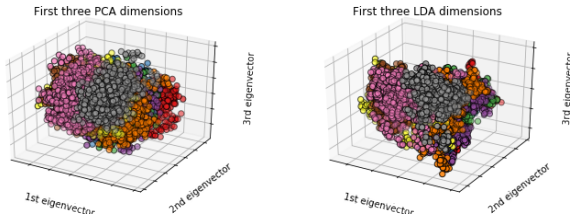

Figure 1. Class separation for PCA and LDA on X_{train} feature vectors mapping first 3 dimensions

1.5. Performance Evaluation

The @RankK measure is used as the primary performance evaluator of distance metrics in this paper[4]. @RankK$_i$ is calculated by using a distance metric to compute a ranklist R_i and

[3]We did not attempt to extract our own feature vectors from images, as time constraints in trying to train a network of similar sophistication would make it difficult to obtain higher quality feature vectors[10]. Furthermore, an extra pre-processing step would be required to standardize (i.e. re-scaling, re-sizing) the images in CUHK03-NP, such that they all have the same pixel dimensions for input into a Residual Neural Network.

[4]The full algorithm is summarised in Appendix II.

then finding the K-Nearest x_j's in $X_{gallery}$ for x_i in X_{query}. If $y_i = y_j$ for any $j \leq K$ where $y_i, y_j \in Y_{query}, Y_{gallery}$, then @RankK$_i$ = 1, else @RankK$_i$ = 0. Finally, RankK = $\frac{1}{N_{query}} \sum_{i=1}^{N_{query}}$ @RankK$_i$.

Mean Average Precision (mAP) [4] is another measure used to evaluate performance. AP is calculated by interpolating 11 values along the precision vs recall curve for a certain query image and summing the area beneath. mAP is the mean of these values over the whole of X_{query}.

2. Baseline Approach

2.1. K-NN Ranklist with Euclidean Distance

The defined problem consists of creating a ranklist of images $\mathcal{G} \in X_{gallery}$, sorted by increasing distance to a given query image $\mathcal{Q} \in X_{query}$. As a first approach we define our baseline method to use the euclidian (L_2) distance (Eq. 1)[5] when creating the ranklist.

$$d(\mathcal{Q}, \mathcal{G}) = L_2(\mathcal{Q}, \mathcal{G}) = \sqrt{\sum_i (\mathcal{Q}_i - \mathcal{G}_i)^2} \qquad (1)$$

This metric essentially models the shortest linear distance between two points in multi-dimensional physical space where each dimension is weighted equally. This means however, that it is not very robust to outliers and will prioritize the dimensions with the highest variance.

For the this distance metric our algorithm gave the following results: @Rank1 = 47.00%, @Rank5 = 66.86%, @Rank10 = 74.93%, mAP[6] = 46.57%.

We observe that the baseline method already achieves fairly accurate results, and is able to determine a correct match @Rank1 for almost half the query images. We once again tie this down to the fact that the given features already represent the data points in a fairly linearly separable way within multi-dimensional space.

2.2. Baseline Method Applied to Training Set

As a further enquiry we applied our baseline method to X_{train}, where each x_i in X_{train} was considered as a query image, and the remaining $N_{train} - 1$ elements of X_{train} as its corresponding gallery images. From this enquiry we obtain an accuracy @Rank1 = 99.82%. This is an interesting result as it reveals that the provided features almost perfectly separate the classes in Euclidean space. This substantiates our original visual analysis with PCA on X_{train}.

We expect that this result stems from the fact that the ResNet-50 used to extract feature vectors contained in X_{train} was also trained on this same X_{train}. An inevitable result of this is that since our training error is already close to zero for euclidean distance, it will most likely be very difficult to improve on this with other distance metrics by validating hyper-parameters on this training set.

[5] i in equation represents dimension indexes of the vectors $\mathcal{Q}$ and $\mathcal{G}$.

[6] Our mAP result slightly diverged from that stated in [4]] , most likely resulting from a differing approximation of the precision vs recall curves.

2.3. k-means with Euclidean Distance

Looking at methods to implement k-means as an additional baseline we observe that in our case we are unable to obtain an exact number for the number of clusters (k) because we must assume that we are unaware of the number of unique identities/labels present in $Y_{gallery}$. Working with this constraint we came up with the idea of implementing clustering before k-NN. Though in a real-world use case this approach seems perfectly acceptable, as to avoid biased results we chose to not consider it as an accurate baseline in the following comparisons and instead detail this improved method in Section 3.4. Furthermore the augmented k-means algorithm would be difficult to classify as a "distance metric" and hence a comparison with other distance metrics may not be sensible.

3. Improved Metrics

3.1. Standard Distance Metrics

As a range of different standard distance metrics, $d(\mathcal{Q}, \mathcal{G})$, we test for Square Euclidean, Manhattan Distance (L_1), Chebyshev Distance (L_∞), Chi Square, Bray Curtis, Canberra, Cosine and Correlation metrics with the same algorithm used for baseline. (Formulas provided in Appendix IV.)

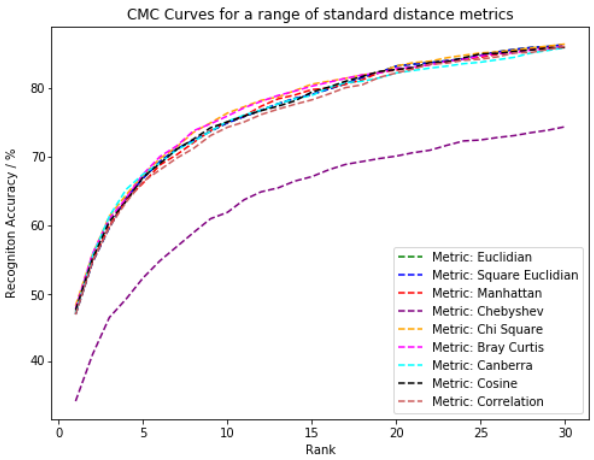

Figure 2. CMC curves for a range of standard distance metrics from @Rank1 to @Rank30

CMC curves and mAP & @Rank1 values are provided in Figure 2 and Table 1. We observe that the results for most of the metrics are very similar. Chebyshev however, appears to have provided a much worse result than the others. This may be due to the fact that it prioritizes only the dimension with the largest differentiation and hence is not a good measure to capture the separation resulting from differences in multiple dimensions.

Looking at the other measures we see that all except for correlation have improved slightly on baseline. The best results are provided by Chi Square, Bray Curtis, Canberra and Cosine. The former three most likely provide an improvement due to the fact that they each inherently weight the dimensions in the distance calculations inversely to the total of the values of those dimensions. Also, as expected, Chi Square and Bray Curtis produce similar results on fairly regularized data [8] . And for Co-

Distance Metric	@Rank1/%	mAP/%
Base. Euclidean	47.00	46.57
Sq. Euclidean	47.00	46.57
Manhattan	47.21	46.78
Chebyshev	34.29	29.43
Chi Square	48.21	47.82
Bray Curtis	47.79	47.52
Canberra	47.21	46.39
Cosine	47.57	47.21
Correlation	46.93	46.73

Table 1. Standard distance metric accuracy scores

sine, which is an angle based measure of dissimilarity we can also conclude that the absolute values of each dimension are less important. Hence, these metrics are more robust to large dimensions as opposed to Euclidean and this most likely helps them more accurately identify certain outliers.

3.2. Mahalanobis Distance Metric Learning

We next apply some distance metric learning techniques to the dataset to see if we are able to improve accuracy. For this we use the Mahalanobis form for distance calculations where the dimensions are weighted by a matrix A (Equation 2[7]).

$$d(\mathcal{Q},\mathcal{G}) = \sqrt{(\mathcal{Q}-\mathcal{G})A(\mathcal{Q}-\mathcal{G})} \tag{2}$$

We consider the two main forms of the Mahalanobis metric: the covariance based version, where the matrix A is defined as the inverse of the covariance matrix of the feature set X_{train}, and the general learnt form where A is learnt through a constrained optimisation problem to achieve the best class separation. For the second method we were limited by the computational capacity available and hence, first preformed PCA on X_{train} with 150 dimensions and then selected a small subset of the data points. The learnt Mahalanobis matrix should ideally minmize the sum of distances between similar classes whilst maximizing the margins between dissimilar classes.

At this point we note the we already achieve around 99% accuracy for the baseline method on X_{train}, indicating an almost optimal class separation in euclidean space. Hence we can hypoyhesize that it will be difficult for Mahalanobis methods to improve on this accuracy, and may essentially learn a metric almost identical to Euclidean.

The results are plotted in Figure 8. The LMNN (Large Margin Nearest Neighbour) is a variant on Mahalanobis and aims to keep same classes within k nearest neighbours and separate other classes by a large margin (k = 3). We see that the covariance based method and LMNN produces unsatisfactory results. More interestingly we see that, as expected, the Mahalanobis matrix appears to have learnt a measure with performance equivalent to the Euclidean distance metric.

3.3. Kernel Methods

Kernels model non-linearity and hence, when integrated with standard distance metrics, may be useful in separating some of the non-linearly separable data. For a given kernel $\mathcal{K}(\mathcal{Q},\mathcal{G}) =< \Phi(\mathcal{Q}), \Phi(\mathcal{G}) >$, the Euclidean distance in high-dimensional Hilbert space can be calculated as in Equation 3.

$$d(\mathcal{Q},\mathcal{G}) = \mathcal{K}(\mathcal{Q},\mathcal{Q}) - 2 * \mathcal{K}(\mathcal{Q},\mathcal{G}) + \mathcal{K}(\mathcal{G},\mathcal{G}) \tag{3}$$

We trial three different types of kernels (Appendix VI): Polynomial ($\gamma = 4.88x10^{-4}$, degree $= 3$), Gaussian/RBF ($\gamma = 4.88x10^{-4}$) and Sigmoid ($\gamma = 4.88x10^{-4}$, c0 $= 1$). From results in Figure 9, we see that RBF and Sigmoid provided almost identical accuracy to baseline, whilst polynomial fell slightly below. Once again we can attribute the lack of improvement to the fact that the data is already fairly linearly separable in the relatively high number of 2048 dimensions. Hence, mapping to an even higher dimensional space with a kernel does not provide much benefit.

3.4. Augmented k-means & K-NN with Euclidean Distance

We now examine our idea of implementing clustering before K-NN as an augmented method for reducing computational complexity, and also imposing extra constraints on the generated ranklist, by first looking at a subset of images assigned to clusters who's centres are closest to the query image rather than ranking the whole of $X_{gallery}$.

We hypothesize that these constraints will result in a different ranklist compared to standard K-NN, as the algorithm will also consider neighbours of its nearest neighbours which would otherwise not be included. This idea can be thought of as intrinsically implementing a similar idea as the improved k-reciprocal re-ranking method [15].

When we consider this approach an issue to bring up is at which stage we preform the k-means clustering: i.e. before or after the identities in $X_{gallery}$ with the same C_i and y_i are removed. The first approach would greatly reduce computational complexity as the k-means algorithm would only need to be computed once, however there exists the question as to whether the effect of the presence of images with the same C_i and y_i in $X_{gallery}$ would bias our results by including more information than should otherwise be considered when computing the centres. We expect this effect to be greater when a higher k is used as cluster centres will more easily be moved by these number of points. Algorithms for both methods are provided in Appendix VIII. Note for both cases, care is taken to ensure images with same C_i and y_i are not included in the accuracy measures.

3.4.1 Results and Execution Time Considerations

From the data, interestingly, we observe that a higher value of k appears to greatly improve accuracy. From Figure 10 we observe, however, that these gains in accuracy appear to be maximum for a k value of around 128 for both pre and post computed versions with respective accuracies @Rank1 of %63.16 and %62.19. We also note that the results of pre-computed k-means don't appear to be significantly biased against the post-computed version. To the contrary, for quite a few values of k the post-computed version produced better results @Rank1 by up to a few percent.

[7]Most algorithms compute the distance without the square root.

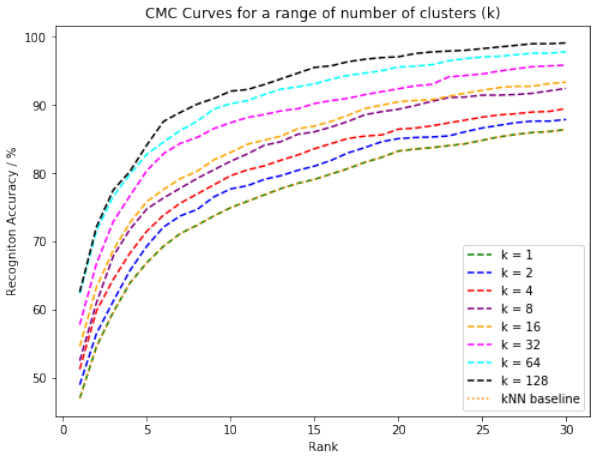

Figure 3. CMC curves for a range of number of clusters (k) from @Rank1 to @Rank30 (Post-computed k-means) (Pre-computed k-means results provided in Figure 10 in Appendix IX)

In terms of computational complexity, as observed in Figure 4, complexity improves on baseline for small values of k in the pre-computed approach. However for the post-computed approach, the complexity is greater than baseline even for small k and explodes for larger values. Hence, this method appears unfeasible for large values of k.

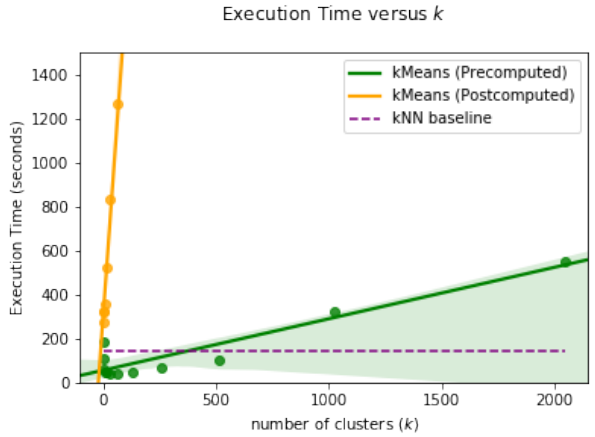

Figure 4. Variation of execution time compared to baseline K-NN for a range of number of clusters (k)

3.5. K-NN with Re-Ranking

As a further method we also aimed to implement simplified versions of the concepts of re-ranking using k-reciprocal neighbours sets, as detailed in paper [15]. However our simplified version provided accuracy only within 1% (47.79% @Rank1 for k=10 reciprocal nearest neighbours) of baseline. A better implementation would have tried to implement better methods to use neighbour sets of neighbours to complement our ranklist with potential matches the K-NN method may have missed.

3.6. Artificial Neural Network (ANN)

We implemented an ANN with fully connected layers based on architectures that take ResNet extracted features as inputs [7]. Our ANN takes triplets of feature vectors at the input layer(6144 Neurons), has two hidden layers with ReLu activation(1024 and 128 Neurons respectively) and 2 softmax output neurons. Each hidden layer undergoes batch normalization in order to speed up training and reduce co-variance shift [5]. Regularization in the form of dropout is also to applied to each hidden layer as a measure to reduce overfitting [12].

We adhered to a training procedure similar to one used in online triplet mining [9]. The first image input a_i is treated as an anchor, the second image p_i has the same label as a_i and is treated as a positive image, the third image n_i has a different label to a_i and is treated as a negative image. $a_i, p_i, n_i \in X_{train}$. The ANN aims to minimize the naive triplet loss:

$$\mathcal{L}(a_i, p_i, n_i) = max(d(a_i, p_i) - d(a_i, n_i) + margin, 0) \quad (4)$$

In order for the ANN to learn effectively, we want (a_i, p_i) to be close together and (a_i, n_i) to be far away in the embedding space. However, we don't want to push the train embeddings of each label to collapse into very small clusters [11]. We choose at random two positive examples of the same class and one negative example, the negative should be farther away than the positive by some margin. This is very similar to the margin used in SVMs, and here we want the clusters of each class to be separated by the margin. From the definition of loss we can define three types of negatives: hard negatives, semi-hard negatives or easy negatives as summarized in Figure 13. Initially, we randomly chose semi-hard negatives based on euclidean distance [3] between a_i and n_i for a given margin, as we thought it could help the network distinguish correct labels better in the semi-hard negative region (Figure 13). However, this did not help improve test accuracy, thus we ended up choosing a random n_i which could be from any of the three negative categories.

We carried out validation by varying various hyper-parameters such as batch-size, training epochs, number of hidden layers and gradient descent optimizers.[8] The optimal hyperparameters we obtained were: batch sizes of 150, 20 training epochs, 2 hidden layers and Adam optimizer. We trained the ANN with these optimal hyper-parameters on the full X_{train}, and following testing we obtained @Rank1 = 38.72%.

4. Conclusion

Concluding our inquiry into distance metric learning we are able to draw a few general conclusions. We saw that it was difficult to improve on Euclidean with any of the standard distance metrics, with the feature set being optimised to X_{train} being the most important factor contributing to this. We achieved some good results with our k-means K-NN augmented solution. Our ANN didn't generalise well to the test data but may still provide promising results if improved. Future work in this area could involve the implementation of a convolutional network to learn optimal features directly from images or an inquiry into the use of Siamese networks[1] for measuring similarity.

[8] Full set of validation results can be found in Appendix XII

References

[1] W. Chen, X. Chen, J. Zhang, and K. Huang. Beyond triplet loss: A deep quadruplet network for person re-identification. 07 2017.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 1512.03385.

[3] A. Hermans, L. Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. *CoRR*, abs/1703.07737, 2017. 1703.07737.

[4] J. Hui. map (mean average precision) for object detection, "Mar 6, " 2018.

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. 1502.03167.

[6] L. I. Kuncheva. *Teaching and research practices in pattern recognition (personal views and experiences)*, pages 63–82. Research and teaching in computing engineering. Universidade da Coruña: Servicio de Publicaciones, 2010.

[7] A. Mahmood, M. Bennamoun, S. An, and F. Sohel. *ResFeats: Residual Network Based Features for Image Classification*. 2017.

[8] B. McCune and J. Grace. *Analysis of Ecological Communities*, volume 289. 01 2002.

[9] O. Moindrot. Triplet loss and online triplet mining in tensorflow, "Mar 19, " 2018.

[10] L. Qi, J. Huo, L. Wang, Y. Shi, and Y. Gao. Maskreid: A mask based deep ranking neural network for person re-identification. *CoRR*, abs/1804.03864, 2018. 1804.03864.

[11] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015. 1503.03832.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[13] M. Taj and A. Cavallaro. Multi-camera scene analysis using an object-centric continuous distribution hidden markov model. In *2007 IEEE International Conference on Image Processing*, volume 4, page 552, 2007. ID: 1.

[14] C. Zhang and L. Huang. Content-based image retrieval using multiple features. *Journal of computing and information technology*, 22(LISS 2013), Nov 10, 2014.

[15] Z. Zhong, L. Zheng, D. Cao, and S. Li. Re-ranking person re-identification with k-reciprocal encoding. *CoRR*, abs/1701.08398, 2017. 1701.08398.

[16] W. Zhou, H. Li, and Q. Tian. Recent advance in content-based image retrieval: A literature survey, Jun 19, 2017.

Appendix I - Distribution of Feature Vectors

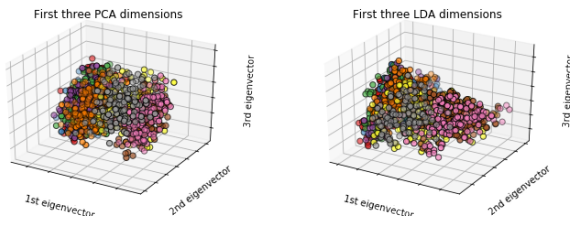

Figure 5. Class separation for PCA and LDA on X_{query} feature vectors mapping first 3 dimensions

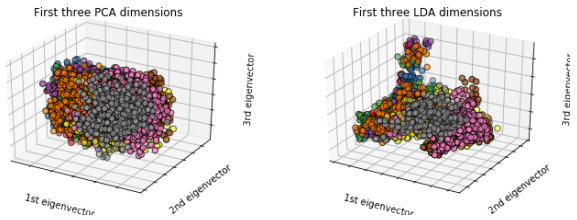

Figure 6. Class separation for PCA and LDA on $X_{gallery}$ feature vectors mapping first 3 dimensions

Appendix II - Algorithm to obtain @RankK measure

Algorithm 1 Procedure to obtain @RankK measure

For x_i in X_{query}: ▷ i $\in [1, N_{query}]$

  For x_j in $X_{gallery}$: ▷ j $\in [1, N_{gallery}]$

    Calculate distance d_{ij} between x_i and x_j

    Find K-Nearest Neighbours (x_j's) to x_i

  If: $y_i = y_j$ for any $j \leq K$ ▷ $y_i \in Y_{query}$

    @RankK$_i = 1$ ▷ $y_j \in Y_{gallery}$

  Else: @RankK$_i = 0$

Compute: RankK $= \frac{1}{N_{query}} \sum_{i=1}^{N_{query}}$ @RankK$_i$

Appendix III - Example Ranklists

Figure 7. Three example ranklists of query images from X_{query} displaying correctly and incorrectly identified matches

Appendix IV - Standard Metric Equations

Euclidean

$$d(\mathcal{Q}, \mathcal{G}) = L_2(\mathcal{Q}, \mathcal{G}) = \sqrt{\sum_i (\mathcal{Q}_i - \mathcal{G}_i)^2} \tag{5}$$

Square Euclidean

$$d(\mathcal{Q}, \mathcal{G}) = \sum_i (\mathcal{Q}_i - \mathcal{G}_i)^2 \tag{6}$$

Manhattan

$$d(\mathcal{Q},\mathcal{G}) = L_1(\mathcal{Q},\mathcal{G}) = \sum_i |\mathcal{Q}_i - \mathcal{G}_i| \tag{7}$$

Chebyshev

$$d(\mathcal{Q},\mathcal{G}) = L_\infty(\mathcal{Q},\mathcal{G}) = \max_i |\mathcal{Q}_i - \mathcal{G}_i| \tag{8}$$

Chi Square

$$d(\mathcal{Q},\mathcal{G}) = \sum_i \frac{(\mathcal{Q}_i - \mathcal{G}_i)^2}{(\mathcal{Q}_i + \mathcal{G}_i)} \tag{9}$$

Bray Curtis

$$d(\mathcal{Q},\mathcal{G}) = \frac{\sum |\mathcal{Q}_i - \mathcal{G}_i|}{\sum |\mathcal{Q}_i + \mathcal{G}_i|} \tag{10}$$

Canberra

$$d(\mathcal{Q},\mathcal{G}) = \sum_i \frac{|\mathcal{Q}_i - \mathcal{G}_i|}{|\mathcal{Q}_i| + |\mathcal{G}_i|} \tag{11}$$

Cosine

$$d(\mathcal{Q},\mathcal{G}) = 1 - \frac{< \mathcal{Q} \cdot \mathcal{G} >}{||\mathcal{Q}||_2 + ||\mathcal{G}||_2} \tag{12}$$

Correlation

$$d(\mathcal{Q},\mathcal{G}) = 1 - \frac{< (\mathcal{Q} - \bar{\mathcal{Q}}) \cdot (\mathcal{G} - \bar{\mathcal{G}}) >}{||(\mathcal{Q} - \bar{\mathcal{Q}})||_2 + ||(\mathcal{G} - \bar{\mathcal{G}})||_2} \tag{13}$$

Where $\bar{\mathcal{Q}}$, $\bar{\mathcal{G}}$ represent the means of $\mathcal{Q}$ and $\mathcal{G}$ respectively.

Appendix V - Results for a Range of Mahalanobis Methods

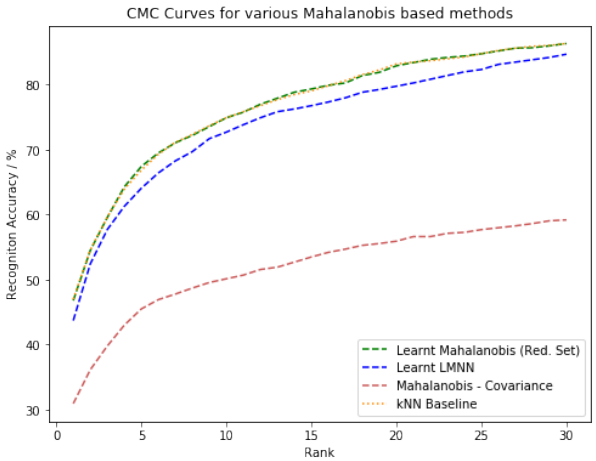

Figure 8. CMC curves for Mahalanobis based methods from @Rank1 to @Rank30

Appendix VI - Kernel Formulas

Polynomial

$$\mathcal{K}(\mathcal{Q},\mathcal{G}) = (\gamma < \mathcal{Q} \cdot \mathcal{G} > + c0)^{degree} \tag{14}$$

RBF - Gaussian

$$\mathcal{K}(\mathcal{Q},\mathcal{G}) = \exp(-\gamma ||\mathcal{Q} - \mathcal{G}||) \tag{15}$$

Sigmoid

$$\mathcal{K}(\mathcal{Q},\mathcal{G}) = \tanh(\gamma < \mathcal{Q} \cdot \mathcal{G} > + c0) \tag{16}$$

Appendix VII - Results for a Range of Kernels

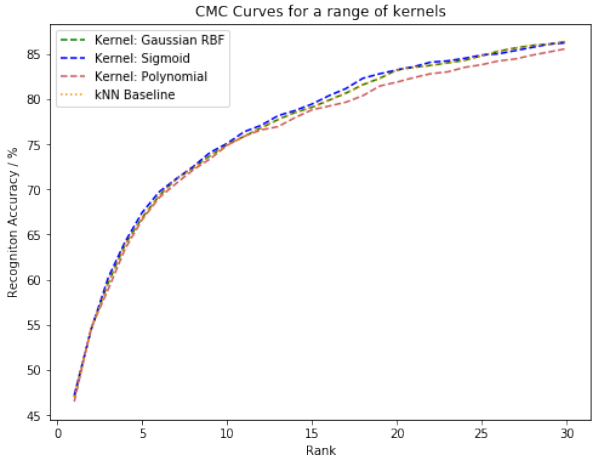

Figure 9. CMC curves for a range of kernels from @Rank1 to @Rank30

Appendix VIII - Augmented k-means with K-NN Algorithm

Algorithm 2 Augmented k-means with K-NN (Post-computed)

For x_i in X_{query}: ▷ i ∈ $[1, N_{query}]$
  Generate $X_{reduced}$ from $X_{gallery}$ by removing images with same C_i and y_i
  Perform k-means on $X_{reduced}$ to generate k clusters C
  Calculate distance d_{im} from x_i to each c_m ▷ $m \in [1, k]$
  Order each c_m in ascending order of d_{im}
  While # considered points < required length of ranklist:
    For x_j in c_m: ▷ c_m = closest unconsidered cluster
      Calculate distance d_{ij} between x_i and x_j
      Find K-Nearest Neighbours (x_j's) to x_i
      Move to next cluster c_{m+1}

Algorithm 3 Augmented k-means with K-NN (Pre-computed)

Perform k-means on $X_{gallery}$ to generate k clusters C
For x_i in X_{query}: ▷ i $\in [1, N_{query}]$
  Calculate distance d_{im} from x_i to each c_m ▷ $m \in [1, k]$
  Order each c_m in ascending order of d_{im}
  While # considered points $<$ required length of ranklist:
    For x_j in c_m: ▷ c_m = closest unconsidered cluster
      Calculate distance d_{ij} between x_i and x_j
      Find K-Nearest Neighbours (x_j's) to x_i
      Move to next cluster c_{m+1}

Appendix IX - Results Augmented k-means (Pre-computed)

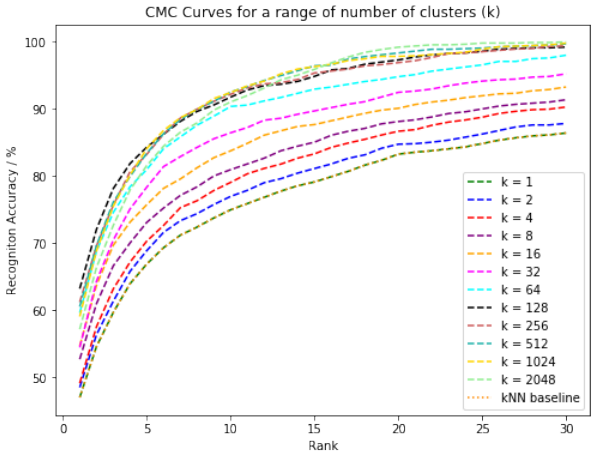

Figure 10. CMC curves for a range of number of clusters (k) from @Rank1 to @Rank30 (Pre-computed k-means)

Appendix X - Execution Time for Augmented k-means

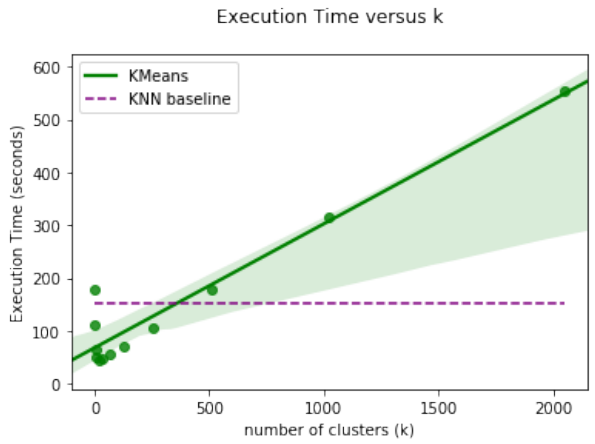

Figure 11. Variation of execution time compared to baseline K-NN for a range of number of clusters (k) (Pre-computed k-means)

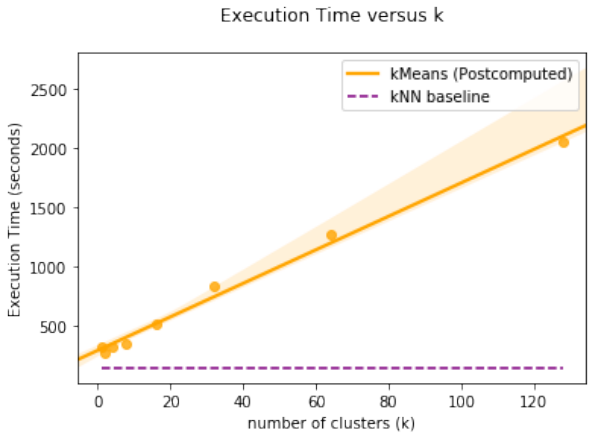

Figure 12. Variation of execution time compared to baseline K-NN for a range of number of clusters (k) (Post-computed k-means)

Appendix XI - Negative Types in Triplet Loss

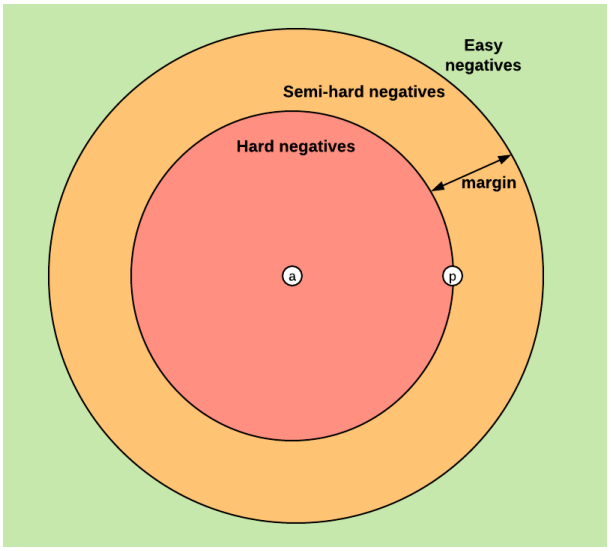

Figure 13. Visualization of the three types of negatives [9]

- Triplets can be categorized into three regions:
  1. Easy Triplets: Triplets which have a loss of 0, because $d(a_i, p_i)$+ margin $< d(a_i, n_i)$
  2. Hard Triplets: Triplets where n_i is closer to a_i than p_i i.e. $d(a_i, n_i) < d(a_i, p_i)$
  3. Semi-hard Triplets: Triplets where n_i is not closer to a_i than p_i, but can still have positive loss: $d(a_i, p_i) < d(a_i, n_i) < d(a_i, p_i)$+ margin

Appendix XII - ANN Validation Results

- Due to computational and time constrains a grid-search based validation method for hyper-parameter tuning was not feasible. Instead, we adopted a method that varies one hyper-parameter, whilst keeping the others constant for each validation run:

Batch Size	@Rank1/%	mAP/%
100	34.9	30.70
125	47.12	36.82
150	49.74	39.53
175	47.67	35.74
200	47.15	39.47

Table 2. Validation accuracies for varying batch sizes

Epochs	@Rank1/%	mAP/%
5	49.74	39.53
10	48.19	36.86
15	50.26	38.42
20	53.37	42.71

Table 3. Validation accuracies for varying training epochs

Optimizer	@Rank1/%	mAP/%
SGD	47.67	37.92
Adam	53.37	42.71

Table 4. Validation accuracies for different optimizers

Hidden Layers	@Rank1/%	mAP/%
1	34.02	24.62
2	48.19	36.86

Table 5. Validation accuracies for different no. of hidden layers

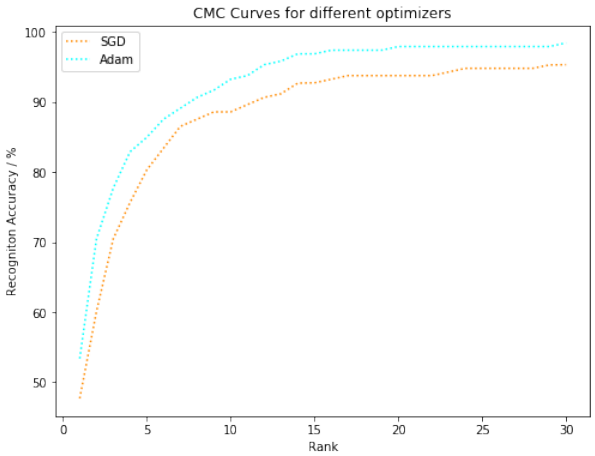

Figure 14. CMC Curves for different Optimizers

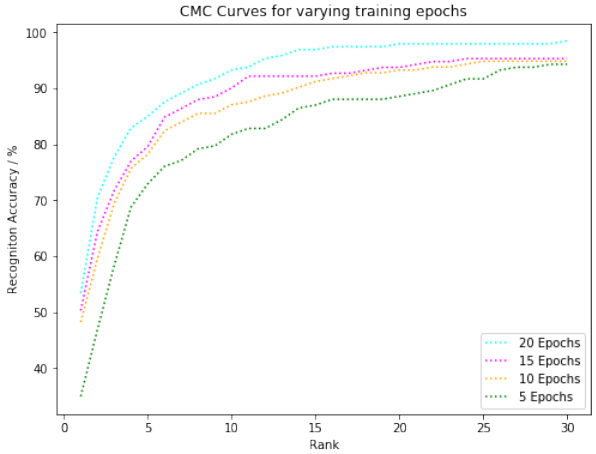

Figure 15. CMC Curves for varying training epochs

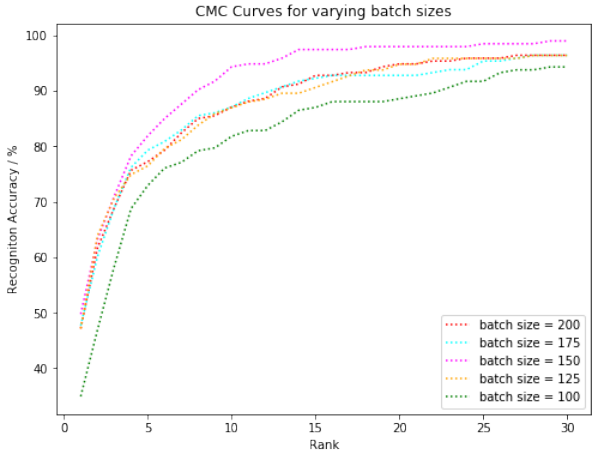

Figure 16. CMC Curves for varying batch sizes

Appendix XIII - Training & Testing Instructions

- **PREREQUISITES**: Ensure you have an installed Python 3 environment on your machine such as the latest Python or Anaconda distribution. Also ensure you have the following dependencies/libraries installed

  1. Tensorflow
  2. Keras
  3. SciPy
  4. NumPy
  5. Pandas
  6. matplotlib
  7. sklearn
  8. metric-learn

- The code for each model used in this report is self contained within its own single Python Script, in other words it trains, tests and outputs the accuracy of a model in the form of graphs and/or terminal outputs.

- If you wish to verify the accuracy of a given model, then proceed by following these instructions:

    1. Download the .zip file of our code repository at this link: **https://drive.google.com/file/d/18r_GArmhLAt3s6pe3U6_C4Q1yBcF6spd/view?usp=sharing**
    2. Unzip the zipped folder and extract its contents into its own new folder on your machine. In this folder you will see two folders: Python Scripts and Jupyter Notebooks.
    3. Copy the contents of the CUHK03-NP data-set provided for this coursework into the Python Scripts folder. It is essential that the cuhk03_new_protocol_config_labeled.mat and feature_data.json files are in the Python Scripts folder, otherwise the respective python scripts of each model will not be able to load the data-set and will fail to run in its entirety.
    4. You can now run any Python Script for any of the models provided in the Python Scripts folder, which will generate all the relevant results reported.

- It is important to note that some files may take a long time to train and test, sometimes over 24 hours depending on the computational power you have at your disposal. All of our models were trained and tested using a Nvidia 840M GPU (lower-end laptop GPU)

- If you wish to run a given model in an interactive Jupyter notebook format then simply add the contents of the CUHK03-NP data-set we were provided with and put it inside the Jupyter Notebooks Folder. Then simply open a Jupyter Notebook and run the code cell by cell. Each model has the same name in both .py and .ipynb formats.

All the files used for training and testing can also be found on our GitHub repository: **https://github.com/RajanPatel97/EE4-68-Pattern-Recognition-CW2**