Retrospective Exam

Arman Afrasiyabi
Computer Vision and Systems Laboratoy (CVSL)
Electrical Engineering and Computer Engineering Department

December 6, 2018

Contents

Chapter 1

Questions of Dr. Christian Gagne

1.1 Question 1: Manifold learning

Answer: Generally, dimension reduction techniques divided into two categories: linear and non-linear. The main assumption in these methods is that data points in the high dimensional space (d) can be represented in low dimensional space (p), where $p < d$. Mathematically, there is an embedding function $f : \mathcal{R}^d \longmapsto \mathcal{R}^p$ which maps the the input high dimension to the output low dimension. Given an example $\mathbf{x}_i$ in the original space $\mathcal{R}^d$ with noise ϵ_i, the goal is to find its map in the low dimension $\mathbf{x}_i^{new} \in \mathcal{R}^p$:

$$x_i^{new} = f(x_i) + \epsilon_i \tag{1.1}$$

Some of the traditional linear dimension reduction technques like principla components analysis (PCA) and multidimensional scaling (MDS) are linear methods with the aim of only pushing the dissimilar point apart (Maaten and Hinton, 2008). Please note that, both of these methods can be extended into non-linear case. However, manifold learning algorithms are based on non-linear dimension reduction approach. The term manifold refers to structure of the data points in the original high dimensional space. Manifold learning algorithms are typically showing superiority over the linear dimension reduction, because their goal is to preserve at least the local non-linear geometries on the original manifold.

The application of manifold learning ranges from pattern recognition and machine learning to data compression and database navigation (Ghodsi, 2006). In the machine learning area, manifold learning algorithms are used for the following three main purposes:

- Finding the low dimensional embedding space.
- Data visualization (mapping the high dimensional original space to 3D or 2D)
- Feature extraction for supervised and semi-supervised learning.
- Clustering by preserving the pairwise distances of the data points.

Base on the global and localpreservation of the manifold structures in the original space, manifold learning algorithms are divided into two groups. First, The global approaches like Isomap (Balasubramanian and Schwartz, 2002), MDS (Kruskal, 1964), Kernel PCA (Mika et al., 1999) aims to preserve all of the global properties. On the other hand, the local approaches such as LLE (Roweis and Saul, 2000), Sammon mapping (Sammon, 1969) and Stochastic Neighbor Embedding (SNE) (Hinton and Roweis, 2003) tries to preserve the local structures of the data in the original space.

1.1.1 What are Kernel PCA, MDS, LLE and t-SNE?

Kernel Principle Component Analysis (PCA)?
Answer: PCA is based on the assumption of representing the data points in the original space (with high dimension) as linear variability. For more information, please see a full description of PCA at the end of chapter 1 (Appendix section). However, this assumption would be incorrect in many real-world problems, where the data have a non-linear manifold (with some curvature) in the original space. In such cases, we can use *kernel* PCA. The idea is to use kernels to do the non-linear mapping from the original space (with high dimensional) to the output space.

Background: Kernel base learning
The idea of the learning algorithms is to change the structure of the data (not space) such that we can end up with linearly separable structure. This is done by adding dimension(s) to the original space to take the advantage of *blase of dimensionality* in which the structure of the data is easier to explore. Please note that added dimension might not good in trams of both computation and statistic [1]. Kernel based methods take the advantage of *blase of dimensionality*. Suppose that we have a mapping function $\phi(.)$ which projects the original input vector to a higher dimension ϕ**(x)**: **x** $\longmapsto$ ϕ**(x)**. Kernels are functions which take two input vectors ($\mathbf{x}_1$, $\mathbf{x}_2$) and return the following output:

$$K_{i,j} = \phi(x_1)^T \phi(x_j) \tag{1.2}$$

There are many kernel functions like linear $K_{ij} = \langle x_i, x_j \rangle$, polynomial $K_{ij} = \big(1 + \langle x_i, x_j \rangle\big)^p$, Gaussian $K_{ij} = exp\big(-||x_i - x_j||^2/2\sigma^2\big)$, and etc.

The linear algorithm called PCA (presented in Appendix section) is used for dimension reduction, and the idea of the kernel is used in kernel-PCA for the non-linear mapping. In kernel-PCA, we change the PCA in such a way that it depends only on the dot product instead of the coordinates of the data points. In other words, we can replace a kernel function with dot products in PCA. For example, we can replace $X^T X$ in the direct PCA (please see appendix section) with $\phi(X)\phi(X)^T$. In this case, the solution of SVD would be:

$$\phi(X) = U\Sigma V^T \tag{1.3}$$

where, the columns of U are the eigenvectors of $\phi(X)\phi(X)^T$. However, the main *disadvantage* of Kernel-PCA over PCA is that the reconstruction is not possible for unknown $\phi(X)$ because, $[:, 1:p]U[:, 1:p]^T\phi(X) \neq [:, 1:p]U[:, 1:p]^T X$. Therefore, the reconstruction steps [2] are not possible. The algorithm of Kernel-PCA is presented in Algo. 1.

$$X_{new} = U[:, 1:p]^T X \tag{1.4}$$

We can back to the original space by $\hat{X} = U[:, 1:p]X_{new}$. Additionally, finding an appropriate kernel with kernel's parameter for a data could be hard in some real-world application.

---

[1] In machine learning and statistics, increasing the dimensionality of the space could cause *curse of dimensionality* which is a well-known problem which the number of data points growth exponentially for learning a well-performed model.

[2] Algorithm 3 (line 4 and 6) at Appendix section)

Algorithm 1 Kernel PCA in direct form

1: $\tilde{X} \leftarrow X - \bar{X}$ ▷ $\bar{X}$ is the mean of X
2: $K \leftarrow \phi(\tilde{X})\phi(\tilde{X})^T$ ▷ K is the kernel matrix
3: $U \leftarrow$ eigenvectors of $\phi(\tilde{X})\phi(\tilde{X})^T$ ▷ Computing the PCs

4: $X_{new} \leftarrow U[:, 1:p]^T \phi(X)$ ▷ encoding the original data
5: $\hat{X} \leftarrow U[:, 1:p]U[:, 1:p]^T \phi(X)$ ▷ decoding the X_{new}; this step is not possible for an unkonwn $\phi(X)$.

Q2.2: Multidimensional Scaling (MDS) and Sammon mapping?
Answer: Like PCA, the original goal of multidimensional scaling (MDS) is to map the input data matrix $X_{d\times n} = [\ \mathbf{x}_1, ..., \mathbf{x}_n]$ (where $\mathbf{x}_1 \in \mathcal{R}^d$) to the lower dimensional $X_{new} \in \mathcal{R}^{p\times n}$ such that $d << p$. Unlike PCA which finds the optimal mapping to apply on the original data for transformation, MDS constructs a new lower-dimensional dataset. MDS aims to preserve the structure of the data as much as possible by keeping the pairwise distance between the data points in the input and output space. To reach its goal, MDS is to minimize the pairwise distances of input data $\mathbf{D}^X$ in the columns of X to the pairwise distance of output data $\mathbf{D}^X_{new}$. Mathematically,

$$\min_{X_{new}} ||\mathbf{D}^{(X)} - \mathbf{D}^{(X_{new})}||^2 \tag{1.5}$$

To minimize the discussed pairwise distance, various methods have been proposed. Here, we will explain *Sammon Mapping*. Sammon et al. (Sammon, 1969) minimized the inter-point differences between corresponding distances in original and output spaces. Let $\mathbf{d}_{ij}^{(X)}$ be the pairwise distance between two data points in input space, and let $\mathbf{d}_{ij}^{(X^{new})}$ be the distance between a pairwise distance of that two examples in the output space. Sammon (1969) used Euclidean as the pairwise distance, and tried to minimize the following error (E):

$$E = \frac{1}{\sum_{i<j} \mathbf{d}_{ij}^{(X^{new})}} \sum_{i<j}^{n} \frac{(\mathbf{d}_{ij}^{(X)} - \mathbf{d}_{ij}^{(X^{new})})}{\mathbf{d}_{ij}^{(X^{new})}} \tag{1.6}$$

Intuitively, the error E is the amount of structural information which is lost in the output space. In fact, the denominator factor $\mathbf{d}_{ij}^{(X^{new})}$ tends to preserve the topology of the input data points at the output space. The output data set X^{new} is first initialized by performing PCA on the original data X which shown to have better performance over the random initialization. Then, the error E in equation 1.6 is minimized and X^{new} is repeatedly updated using gradient descent. Here, we take the gradient of E w.r.t X^{new}.

Typically, MDS algorithms are much slower compared to PCA, and they requires learning rate setting and specifying the number of iterations. The convergence of gradient descent should be determined experimentally, and the convergence is not guaranteed. Many implementatiosns uses PCA as an initialization point of the algorithm. The computational cost of Sammon mapping is $O(N^2)$ which is very heavy. Recently, some attempts such as (Wang, Fang, and Wang, 2016) aims to solve the computational cost of sammon mapping. Additionally, Sammon mapping puts importance on small distances, and decreases the importance of the large distance.

Q2.3: Locally Linear Embedding (LLE)?
Answer: Locally linear embedding (LLE) is another non-linear algorithm for computing the low dimensional data X^{new} from the input high dimensional input data X. Like the discussed methods, LLE tries to map the data set with d dimension into a global coordinate system of p dimension. LLE aims to preserve the neighborhood embedding of X. The assumption is that a data point $\mathbf{x}_i$ and its k neighbors lie on a local linear patch of all data's manifold. Algorithm 2 represent LLE procedure for dimension reduction. In this procedure, we first build a k-nearest neighbour graph (kNN-graph). Next, we compute the weights $w_i = [w_{i,1}, ..., w_{i,k}]_{k\times 1}$ between $\mathbf{x}_i$ and its k neighouring data points $N_i = [\mathbf{x}_1, ...,\mathbf{x}_n]_{d\times k}$ by minimizing the following objective function:

$$\min_{w} \sum_{i=1}^{n} ||\mathbf{x}_i - \sum_{j=1}^{k} w_{ij}\mathbf{x}_{N_i(j)}||^2 \tag{1.7}$$

Finally, LLE transforms the high-dimension to the lower one by minimizing the following objective function:

$$\min_{\mathbf{x}^{new}} \sum_{i=1}^{n} ||\mathbf{x}_i^{new} - \sum_{j=1}^{k} w_{ij}\mathbf{x}_{N_i(j)}^{new}||^2 \tag{1.8}$$

Algorithm 2 LLE based on kNN-graph

1: $G \leftarrow$ kNN-graph ▷ Specifying the neighbours of $\mathbf{x}_i$
2: $w_i \leftarrow$ weights between $\mathbf{x}_i$ and its neighbourhood in G
3: ▷ Compute the best weights
4: $X^{new} \leftarrow$ lower dimensional data version of X ▷ Can be reconstructed by W

Let $\sum_{j=1}^{k} w_{ij}\mathbf{x}_{N_i(j)} = N_i\ \mathbf{w}_i$. In this case, the equation 1.7 turns to $||\mathbf{x}_i - N_i\ \mathbf{w}_i||^2$. Additionally, let $e = [1, ..., 1]_{k\times 1}$ be a vector, and let $[\mathbf{x}_i 1, ..., \mathbf{x}_i k]_{d\times k} = \mathbf{x}_i\ \mathbf{e}^T$. In LLE, we want $\mathbf{x}_i = \mathbf{x}_i e^T\ \mathbf{w}_i$. Using these notation we can rewire Eq. 1.7 in $||\mathbf{x}_i e^T\ \mathbf{w}_i - N_i||^2$, and is openning would be:

$$\min\left(\mathbf{w}_i^T(\mathbf{x}_i e^T - N_i)^T(\mathbf{x}_i e^T - N_i)\mathbf{w}_i\right) \tag{1.9}$$

$G = \mathbf{w}_i^T(\mathbf{x}_i e^T - N_i)^T(\mathbf{x}_i\ \mathbf{e}^T - N_i)$ is known and called *Gram* matrix. Therefore, we want to minimize the following objective function:

$$\min \mathbf{w}_i^T G \mathbf{w}_i \tag{1.10}$$

without any constrain, it is clear that $\mathbf{w}_i^T = 0$ is an obvious solution to this problem. Therefore, we need the constrain. In LLE, $\sum_{j=1}^{k} w_{ij} = 1$ is used as constrain. Finally, the objective function would be:

$$\begin{aligned} &\min \mathbf{w}_i^T G \mathbf{w}_i \\ &\text{s.t. } e^T \mathbf{w}_i = 1 \end{aligned} \tag{1.11}$$

The Lagrangian of this objective function would be: $L(\mathbf{w}_i, \lambda) = \mathbf{w}_i^T G \mathbf{w}_i - \lambda \mathbf{e}^T \mathbf{w}_i - 1$, and the derivative of the Lagrangian w.r.t w

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}_i} &= 2G\mathbf{w}_i - \lambda e = 0 \\ 2G\mathbf{w}_i &= \lambda e \\ G\mathbf{w}_i &= \frac{\lambda}{2} e \end{aligned} \tag{1.12}$$

It seems that we still can not solve $\mathbf{w}_i$ because λ is unknown. Meaning that any wrong λ can scale the resulted $\mathbf{w}_i$. In other to solve this issue, we should normalize $\mathbf{w}_i$ after considering any arbitrary λ. LLE is problematic in the case of noisy data, and it introduces a new parameter k. Finding the best k in high dimensional data is not a trivial task.

Q2.3: t-distributed stochastic neighbor embedding (tSNE)?

t-distributed stochastic neighbor embedding (tSNE) is built upon stochastic neighbor embedding (Maaten and Hinton, 2008) (SNE). Before tSNE, let see SEN. Suppose $X = [\mathbf{x}_1, ..., \mathbf{x}_d]_{d \times n}$ be the original data points at $\mathcal{R}^{d \times n}$ space, and let $X^{new} = [\mathbf{x}_1^{new}, ..., \mathbf{x}_d^{new}]_{p \times n}$ be the output dataset. In SNE, we first convert the pairwise distances between the points to probabilities. For any x_i and x_j in the input space, we define probability in the following form:

$$P_{j|i} = \frac{\frac{e^{-|x_i - x_j|^2}}{2\sigma_i^2}}{\sum_{k \neq i} \frac{e^{-|x_i - x_k|^2}}{2\sigma_i^2}} \tag{1.13}$$

where, σ_i is the variance of the fitted Gaussian over the data points. $P_{j|i}$ is the probability that the point i choose point j as its neighbour. Please note that $P_{i|i} = 1$ under Eq. 1.13. We set $P_{i|i} = 0$, because we do not want to choose $\mathbf{x}_i$ as the neighbour of it self. Additionally, please note that $P_{i|j} \neq P_{j|i}$. Please note that $P_{i|j}$ is not a symmetric measurement due to different σ_i. These conditions are necessary to have a metric criteria. For more information, please see the Appendix section. On the other hand, in the low/output space, we define the following probability between pairs of the points:

$$Q_{j|i} = \frac{e^{-|x_i^{new} - x_j^{new}|^2}}{\sum_{k \neq i} e^{-|x_i^{new} - x_k^{new}|^2}} \tag{1.14}$$

where, $q_{j|i}$ is the probability between the data points in the output/low space. Please note that we consider the same variance in output space, and $\sigma = \frac{1}{\sqrt{2\pi}}$. Now, the following KL divergence cost function is defined to minimize the distance between $P_{i|j}$ and $Q_{j|i}$:

$$\min_{x_i^{new}, x_j^{new}} KL(P||Q) = \sum_{ij} P_{j|i} log \frac{P_{j|i}}{Q_{j|i}} \tag{1.15}$$

This cost function has a problem. The problem is related to the property of KL divergence which is not symmetric. Suppose that $\mathbf{x}_i$ is close to $\mathbf{x}_j$ in the original ($P_{i|j}$ is high). Let have a mistake and model $Q_{i|j}$ to be low. Meaning that we could not preserve the manifold. In this case, the cost would be high, and we will not have a

problem. However, the problem happens when $P_{i|j}$ is low ($\mathbf{x}_i$ is far to $\mathbf{x}_j$), we model it with high $Q_{i|j}$. In this case, the cost would be low, the reverse of what we need!

So far, we presented the idea of SNE. In tSNE, a new version of SNE, the authors tried to make $P_{i|j}$ symmetry by considering the joint distribution P_{ij} instead of the conditional distribution proposed by SEN.

$$P_{ij} = \frac{\frac{e^{-|x_i - x_j|^2}}{2\sigma^2}}{\sum_{k \neq i} \frac{e^{-|x_i - x_k|^2}}{2\sigma^2}} \tag{1.16}$$

Please note that σ^2 does not depend on the $\mathbf{x}_i$ anymore, unlike SNE in Eq. 1.13. In fact, the use of fixed σ in tSNE reduces the computation complexity related finding the best σ_i in SNE. In tSNE, authors also changed the distribution in the output space to address the problem of *crowding the points* which also exists in all of the discussed manifold learning algorithms. Crowding the points happens because the volume of the low dimensional space (at the output of the model) is lower than the original high dimensional space. For example, the volume of a sphere full of data points in 3D is much more than a circle in the output 2D space. Therefore, mapping the points by only the pairwise distance might fail to keep the structure of the data for the visualization. To address this issue, the authors used t-distribution, which has long tails. Unlike Gaussian distribution, t-distribution let to cover and capture more volume by its long tails. Therefore, the probability in the output space is defined by:

$$Q_{ij} = \frac{1/|1 + x_i^{new} - x_j^{new}|^2}{\sum_{k \neq j} 1/|1 + x_i^{new} - x_k^{new}|^2} \tag{1.17}$$

Finally, the kl divergences in the form of Eq. 1.15 is minimized for training purpose. tSNE maintains the local structure in data. It works very well in the cases where the classes are linearly separable. However, it might fail to work properly in the case of having many overlapping classes.

1.1.2 Implementation and visualization?

In this question, a pre-trained AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) on ImageNet model is fine-tuned using CIFAR-10 data. Before fine-tuning, the last FC layer of the pre-trained AlexNet is replaced with a new FC layer with 10 neurons. For fine-tuning, there are many options: only fine-tuning the last layer and fixing the other, or fine-tuning some of the last layers, or performing fine-tuning on all of the layers. We preformed the last one.

Figure 1.1 represents the results of four different manifold learning (in columns) algorithm with different parameters. These results are obtained by choosing 600 examples on the test set of CIFAR10. Different classes are represented by different colored shapes. The first column presents the results of Kernel PCA with rbf, cosine, and sigmoid kernels shown in row1, row2, and row3 respectively. As the results of PCA shows, choosing kernel is very important, and the results can vary based on the dataset. The second column of the figure shows the results of LLE with 20, 10 and 5 as the number of neighbors in row1, row2, and row3 respectively. In LLE, finding the number of neighbors has a huge impact on the visualization. The third column of the figure illustrates the results of MDS. In this column, the first row is the result of MDS after 100 iterations. The second and third columns are the results of MDS after 200 and 500 iterations. Compared to Kernel PCA and LLE, the results of MDS are better, especially in high iterations. However, we will see that MDS consumed very

Manifold learning Algorithms			
Kernel PCA	LLE	MDS	tSNE
0.59	3.08	5.44	15.91
0.06	2.89	326.79	16.14
0.07	2.69	1084.56	15.37

TABLE 1.1: Computation time (in second) of the different methods with different parameters. The times corresponds to the table 1.1

more time compared to the other methods. The last column represents the results of tSNE. The rows of the tSNE column are repeating the algorithm. Here, our aim is to see if the algorithm is stable and see the different consumed times.

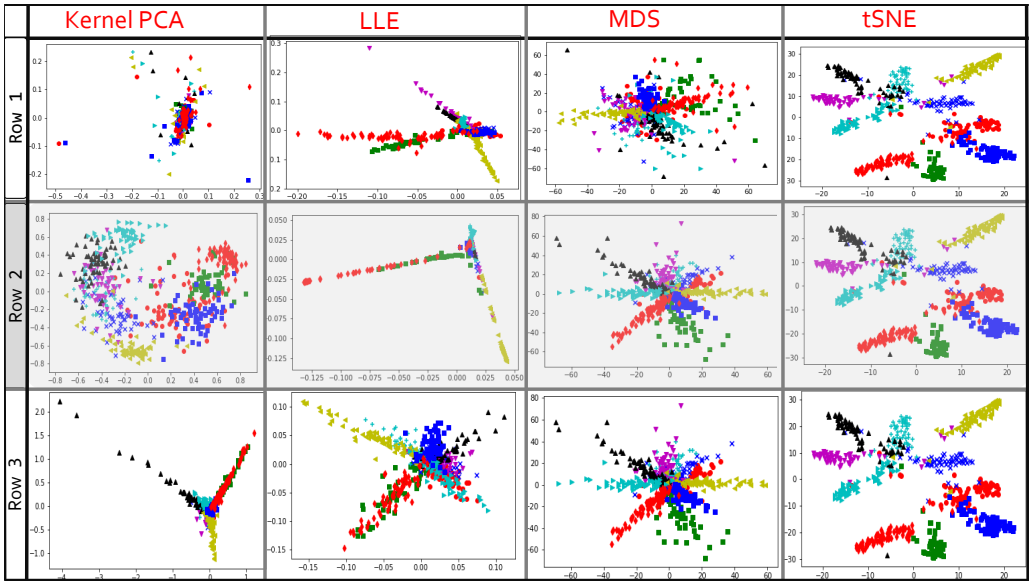


FIGURE 1.1: Visualization results of fine-tuned AlexNet (at the last layer before last FC) on CIFAR10. Each column shows different manifold learning algorithm. Each column corresponds to different parameters except in tSNE.

Table 1.1 present the amount of time each of the algorithms consumed. the rows of the table correspond to the settings in the Figure 1.1 which is discussed in the last paragraph. As the table shows the fastest algorithm is Kernel PCA, and the slowest one is MDS. In fact, these are discussed in the previous question.

1.1.3 Supervised manifold learning?

Answer: Generally, almost all of the manifold learning algorithms have been proposed in an unsupervised way. However, as discussed in part 1 of this question, one of the applications manifold learning algorithms is in semi-supervised and supervised learning. In some cases, researchers used manifold learning algorithms to tackle the curse of dimensionality problem (Chen et al., 2018). In the following paragraphs, one semi-supervised and one supervise example of manifold learning is covered.

Yang et al. (2006) proposed semi-supervised manifold learning (SSML). They used of m number of label known data $X^k = [x_1^k, ..., x_m^k]_{d \times m}$ to construct the the

low-dimensional coordinates. Based on this coordinates, Yang et al. (2006) calculated the low-dimensional coordinates of label unknown data $X^u = [x_1^u, ..., x_n^u]_{d \times n}$. The method consists of three steps: First, they finding the local neighborhoods of X^u. Next, they extracting local geometry of using a local optimization method like LLE (please see part 2 of this question). Finally, they construct a model based on a semi-supervised optimization method in the following form:

$$\sum_{i=1}^{n+m} ||y_i^u - \sum_{x_j \in N_i} w_{ji} y_j^u||^2 + \beta \sum_{i=1}^{m} ||y_i^u - y_i^k||^2 \tag{1.18}$$

where N_i is the set of the neighboring data-points of y_i^u, β is a regularization parameter, and y_i^u and y_i^k are the unknown and known data points in the output space corresponding to x_i^u and x_i^k in the original input space, and w_{ji} is the weight between these points.

In another study, Raducanu and Dornaika (2012) proposed a supervised version of a manifold learning algorithm called Supervised Laplacian Eigenmaps (S-LE). Their algorithm was based on Laplacian Eigenmaps (LE) of Belkin and Niyogi (2003). In LE, we first compute the neighborhood graph on the examples in the original space. The weight of each edge between $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined by symmetric affinity function like Gaussian:

$$w_{ij} = exp(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{\beta}) \tag{1.19}$$

where β is a normalization factor. Then, we transform the input space to the output lower dimensional one by minimizing $1/2 \sum_{i,j} ||x_i^{new} - x_j^{new}||^2$. In Supervised Laplacian Eigenmaps (S-LE), however, Raducanu and Dornaika (2012) spit the graph into two components: the between-class graph G_b and the within-class graph G_w. Let l_i be the label of $\mathbf{x}_i$. Then, the average similarity of the $\mathbf{x}_i$ is computed with the other points in the space:

$$Avg_{sim} = \sum_{k-1}^{N} exp(\frac{-||\mathbf{x}_i - \mathbf{x}_k||^2}{\beta}) \tag{1.20}$$

Next, we build two subsets N^{sim} and N^{dif} contain the neighbors sharing the similar and different to l_i (label of $\mathbf{x}_i$) respectively. These subsets have the following forms:

$$\begin{aligned} N^{sim} &= \{\mathbf{x}_j | l_j = l_i, \ \ exp(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{\beta}) > Avg_{sim}\} \\ N^{dif} &= \{\mathbf{x}_j | l_j \neq l_i, \ \ exp(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{\beta}) > Avg_{sim}\} \end{aligned} \tag{1.21}$$

Finally, the label of the new data point will decided using N^{sim} and N^{sim}. For more details please see the original paper Raducanu and Dornaika (2012).

1.1.4 Relate Manifold learning to prototypical network?

Answer: Snell, Swersky, and Zemel (2017) proposed prototypical network (PN), for k shot learning problem. They used k number examples per N classes. PN maps all k examples in each of the N classes from the input space to single prototype: $\mu_c = \frac{1}{k} \sum_{(x_i, y_i) \in S} f_\theta(x_i)$, where k is the number of support examples (shots) that are used in each iteration of the training, and $f_\theta(.)$ is the mapping function from the

input to the embedding space. Typically, the idea of the prototypical network (PN)

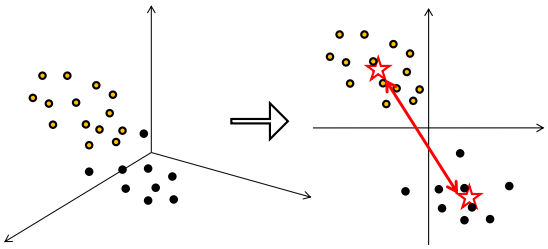


FIGURE 1.2: Prototype based supervised manifold learning.

contradicts with the idea of the manifold learning algorithm. In manifold learning algorithms, we would like to preserve the structure of the data points in the input space and the output space. On the other hand, in PN, our aim is to shrink all of the points in the output space in a single point at the output space.

However, the key question here is: *Does it beneficial to adapt a manifold learning algorithm to prototype based classification?* In fact, the idea of local based manifold learning will have advantages for few-shot learning cases. In the locally based algorithms such as tSNE and LLE, the aim is to keep the local structure of the data. This could be very critical for some distance based few shot learning algorithms. The reason is the low number of examples which might not cover all of the data distribution. In this case, pushing all of the examples towards a single point (like PN network) is not a good idea. Therefore, it would be better to preserve the neighboring structure (local manifold) the data.

One interesting idea would be keeping the within-class structure and local class based manifold as shown in Figure 1.2. This will have the advantage of covering more volume of a space per class in the resulted in low dimensional space. Obviously, it would be better than destroying the within-class manifold and pushing all of the points toward a single point. For example, suppose that we want to make a supervisor version of t-distributed stochastic neighbor embedding (tSNE). We propose, first, computing the probability distribution of the points in the input space P^c_{ij} and output space Q^c_{ij} for each class c in the following form:

$$P^c_{ij} = \frac{\frac{e^{-|x_i - x_j|^2}}{2\sigma^2}}{\sum_{k \neq i} \frac{e^{-|x_i - x_k|^2}}{2\sigma^2}} \tag{1.22}$$

$$Q^c_{ij} = \frac{1/|1 + x^{new}_i - x^{new}_j|^2}{\sum_{k \neq j} 1/|1 + x^{new}_i - x^{new}_k|^2} \tag{1.23}$$

Then, we design the following cost function to preserve the within class manifold and increase the distance between the prototypes (mean) of the class by adding a second margin based optimization function:

$$\sum_{c=1}^{C} kl(P^c_{ij}||Q^c_{ij}) + \beta\Big(\frac{1}{|C-1|} \sum_{\substack{\forall c' \in C \\ c' \neq c}} max\big(0, m - ||\mu^c - \mu^{c'}||_2^2\big)\Big) \tag{1.24}$$

where, where μ^c is the proto of class c, and β is trade-off parameter, and m is the margin between the classes, and c' is the complement classes of c in class set C.

1.2 Question 2: Local and distributed representation and metric learning

1.2.1 In your own words, provide a synthesis of the comparison between local models vs. distributed representations.

Answer: The simplest way of object/input representation is called **local representation**. A well-known example of this type of representation would be representing the different items/objects with a *one-hot* vector. Suppose that we have set of four locations in the memory and set of four characters: {I, H, E, F} which is illustrated in Fig. 1.3 (a). In the local representation, the memory is saturated with only these four characters. In neural networks, the local representation can be dedicating one neuron to each item [3]. Typically, local representations are easy to understand and code by hand. They are also easy in terms of learning. However, they are very inefficient with most of the real world datasets. Because most of these objects in the real world have componential structures. In our example, the character "I" can be constructed out of all three other characters. Additionally, we can not store a new character because all of the memory slots are used.

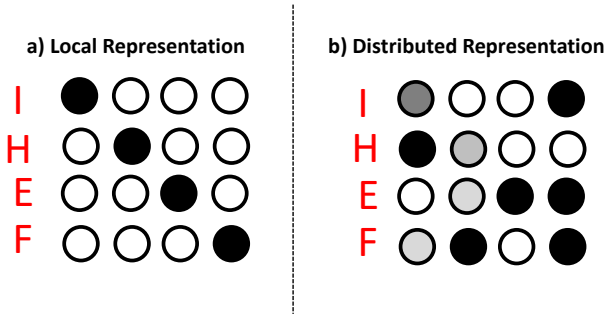


FIGURE 1.3: Local representation vs Distributed representation.

In contrast to *local representation*, we can represent N different items with $log_2 N$ bits in **distributed representation** which is shown in Fig. 1.3 (b). Here, a single concept is represented by more than only one memory in our discussed example, and each neuron is used to represent many concepts. In the neural network, one neuron can be fired for more than one objects.

Some of the clustering algorithms are considered as non-distributed representation learning because the clusters are considered as mutually exclusive. On the other hand, some dimension reduction algorithms like ICA (Pearlmutter and Parra, 1996) and PCA (Hotelling, 1933), and model deep learning algorithms like autoencoder (Vincent et al., 2008) are considered as distributed representation learning.

1.2.2 Determine (with arguments) whether prototypical networks are local models, distributed representations or a mix of both.

Answer: In traditional transfer learning algorithm using deep learning, we first train a network on a large labeled data like ImageNet (Krizhevsky, Sutskever, and Hinton, 2012). Then, we replace the last fully connected layer (called the classification layer) with the new classification layer for fine tuning on a new dataset. This approach has two problems: I) gradient descent needs many examples in meta-layer

[3] http://www.cs.toronto.edu/ bonner/courses/2014s/csc321/lectures/lec5.pdf

for learning, II) fine-tuning can destroy the learned weights in the base layer and causes **catastrophic forgetting** (Goodfellow et al., 2013).

The prototypical network is proposed by Snell, Swersky, and Zemel (2017) for few-shot learning in the meta-learning framework. In fact, the prototypical network is adapting the neural network to the meta-learning framework where there is two level of learning: base level and meta level. Meta-learning is the main framework for the few shot learning in current literature. The reason for this interest is that we can transfer knowledge from base level to a meta-level in a very wise way compared to traditional transfer learning methods. The idea is to do not usefully connected layer (classification layer) none of base and meta-layers. Instead, Snell, Swersky, and Zemel (2017) build distance based metric space. Meaning that the last layer of the network is independent of types of classes. For instance, if we put only a few examples of 5 different classes, the last layer of the network will try to put same class examples in locally near region.

Let us back to the question: *determine if the prototypical network (PN) is a local, or distributed representation, or both*. It is obvious that layers of the prototypical network are distributed representation. In fact, as the name refers, PN is a neural network, and the neurons in the layers of PN can be fired for different objects or items. even the last convolution layer in PN has distributed representation.

As discussed in the previous question, clustering algorithms are non-distributing representation. Additionally, PN similar to clustering algorithms at the prototype space in the last layer of PN. Therefore, the question is: *are the prototypes of classes local representation?* From the prototype perspective, we can say that prototypes are *local representation* because "clustering" form of prototypes causes to loss of information about the data points in the embedding space. This property of clustering algorithms is also discussed by Bengio et al. (2009).

1.2.3 Relate proto. net. with approaches proposed for metric learning.

Answer: In metric learning, we are trying to learn a new metric by learning a matrix A in the following form:

$$|x_i - x_j|_A^2 = (x_i - x_j)^T A (x_i - x_j) \tag{1.25}$$

which A is any *positive semi-definite matrix*, and x_i and x_j are the data points. In other words, $|x_i - x_j|_A^2$ is a distance as long as A is positive semi-definite matrix. In metric learning algorithms, researchers are using objective function to find A. For examples if our goal is to minimize the distance between classes, the following objective function could be designed:

$$\min \sum_{\forall x_i, x_j \in c} |x_i - x_j|_A^2 - \sum_{\substack{\forall x_i, x_j \in c' \\ c' \neq c}} |x_i - x_j|_A^2 \tag{1.26}$$

where the first part minimizes the overall distances in the same class, and the second part is maximizing the between classes distances. The constraint is that A have to positive semi-defined matrix.

For non linear metric learning, we can use kernel based models. The other way of non-linear metric learning is using parameters approaches like neural network (f_θ). In this form, we would like to learn $||f_\theta(x_i) - f_\theta(x_j)||_2$. Kulis et al. (2013) covers two nonlinear metrics based neural networks. Chopra, Hadsell, and LeCun (2005) used

convolutional neural networks f_θ as the mapping function. They used the following objective function for metric learning purpose:

$$\sum_{(i,j,k)\in\mathcal{R}} \lambda_1 d_{i,j} + \lambda_2 \exp(-\lambda_3 \sqrt{d_{i,k}}) \tag{1.27}$$

where, d_{ij} is the learned squared distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ examples of same class, and d_{ij} is the distance between $\mathbf{x}_i$ and examples of other classes $\mathbf{x}_k$.

Salakhutdinov and Hinton (2007) proposed an extension of Neighbourhood Components Analysis (NCA) to a nonlinear case using a neural network. In this study, the Mahalanobis distance is replaced with a nonlinear distance function in the form of a multi-layer neural network like (Chopra, Hadsell, and LeCun, 2005). In NCA, the transformation matrix A is expressed in the following probability form:

$$p^A(j|i) = \frac{\exp(-d(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k\neq i} \exp(-d(\mathbf{x}_i, \mathbf{x}_k))}. \tag{1.28}$$

Generally, metric based few-shot learning algorithms such as (Snell, Swersky, and Zemel, 2017) and (Vinyals et al., 2016) are inspired from metric learning algorithms. Here, the author first computed the prototype for class c, $\mu_c = \frac{1}{k}\sum_{(x_i,y_i)\in S} f_\theta(x_i)$, where, k is the number of support examples (shots) that are used in each iteration of the training, and $f_\theta(.)$ is the mapping function from the input to the embedding space. Then, we compute the probability of a new input example (query) x_q to be assigned to the class c is computed as by following softmax layer:

$$p_\theta(y_c|x_q) = \frac{exp\big(-d(f_\theta(x_q), \mu_c)\big)}{\sum_{c'} exp\big(-d(f_\theta(x_q), \mu'_c)\big)} \tag{1.29}$$

where, d is distance metric.

We believe that prototypical network does not preserve the within class structure. Therefore, there is no direct relation between manifold learning algorithms and prototypical network. However, it seems that the initial intution of prototypical network might come form the study of Chopra, Hadsell, and LeCun (2005). However, as we discussed in the previous section, we think that prototypical network loss the information of data points in the output space. Therefore, it would be an interesting project to extending the work of (Chopra, Hadsell, and LeCun, 2005) to few-shot learning under meta learning framework. More specifically, we can use Eq. 1.28 instead of computing the prototype in Eq. 1.29.

1.3 Appendix I: Principal Component Analysis (PCA)

PCA is a dimension reduction or feature extraction that transforms the data from higher d-dimensional space into a new coordinate system of dimension p, where $p \leq d$. In pure PCA, the assumption is that the data points in the original (high) space are linear, and there is no non-linear manifold of the points. The aim of PCA is to find a subspace (which is a low dimension) of the original high dimensional space. For example, suppose that we have the data points in 2D which are also linearly spread on a subspace. In this case, the goal of PCA is to find the mapping of the projected points on a single line and complete the mapping from 2D to 1D.

Generally, we want to map $x \in \mathcal{R}^d$ to $y \in \mathcal{R}^p$ where $p << d$. In the discussed example, we want to find the *principle components* (shown by $\mathbf{u}_1$ and $\mathbf{u}_2$ in the figure)

which data can be mapped on them. Please note that the components are orthogonal. Therefore, one way of interpreting PCA is to rotate the coordinates of the original space such that the data can represent by less number of output coordinates (principal components). From another perspective, we want to find such principle components that the variations of the **projected** data are as maximum as the original space. In our example, data on $\mathbf{u}_1$ has maximum variation. More formally, for a given set of data vectors x_i, $i \in 1, ..., n$ the p principal axes are those orthonormal axes onto which the variance retained under protection is maximal. Principle components (PCs) are denoted by $\mathbf{u}_1$, $\mathbf{u}_2$, ..., $\mathbf{u}_d$. Generally, we used a subset of p PCs $\mathbf{u}_1$, $\mathbf{u}_2$, ..., $\mathbf{u}_p$. In the following section, we will see how to derive the principle components.

Suppose that we have $X = \left[\mathbf{x}_1, ..., \mathbf{x}_n \right]_{d \times n}$, where $\mathbf{x}_i \in \mathcal{R}^d$. For the first PC, we want to find $\mathbf{u}_1^T X$, where $\mathbf{u}_1 \in \mathcal{R}^{1 \times d}$ and $X \in \mathcal{R}^{d \times n}$. Note the result of $\mathbf{u}_1^T X$ would be projections of the points on $\mathbf{u}_1$. Backing to the goal of PCA, we want $var\left(\mathbf{u}_1^T X\right)$ be maximum:

$$\max_{\mathbf{u}_1} var\left(\mathbf{u}_1^T X\right) \tag{1.30}$$

Suppose that the data points in X have $S \in \mathcal{R}^{d \times d}$ covariance matrix. Here, equation 1.30 will be as follow

$$\max_{\mathbf{u}_1} var\left(\mathbf{u}_1^T X\right) = \max_{\mathbf{u}_1} \mathbf{u}_1^T S \mathbf{u}_1 \tag{1.31}$$

In fact, S is the scalar, and we have an optimization problem. Typically, we need to take derivative and set it to zero in order to maximize equation 1.31. However, this is a quadratic function and has U shape, and there is no upper-bound. Therefore, we cannot find any maximum for this function. Intuitively, multiplying $\mathbf{u}_1$ by a constant S does not have an upper bound.

One way to solve the discussed ill-definition is to adding a constrain to equation 1.31. On contain would be fixing the length of the $\mathbf{u}_1$ to have the length of 1: $\mathbf{u}_1^T \mathbf{u}_1 = 1$. According to this contain, we want to care about finding the direction of the vector instead of finding its length.

$$\max_{\mathbf{u}_1} \mathbf{u}_1^T S \mathbf{u}_1 \quad s.t. \mathbf{u}_1^T \mathbf{u}_1 = 1 \tag{1.32}$$

This technique of putting constant is called Lagrange multiplier. In this case, we should first find the Lagrangian, then we should maximize the equation 1.33

$$L(\mathbf{u}_1, \lambda) = \mathbf{u}_1^T S \mathbf{u}_1 - \lambda_1 (\mathbf{u}_1^T \mathbf{u}_1 - 1) \tag{1.33}$$

where, λ_1 is a dual variable for first PC.

Background: Lagrange Multipliers
If we have the following optimization problem:

$$\max f(x, y) \quad s.t. g(x, y) = c \tag{1.34}$$

The optimization problem in the form of 1.35 will have the the maximum value in the tangent of $g(x,y)$ and $f(x,y)$.

$$\begin{aligned} L(f,\lambda) &= f(x,y) - \lambda\big(g(x,y) - c\big) \\ \nabla f(x,y) - \lambda \nabla g(x,y) &= 0 \\ \nabla f(x,y) &= \lambda \nabla g(x,y) \end{aligned} \tag{1.35}$$

Now, we can take the derivative of the original problem in the following form:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{u}_1} &= 2S\mathbf{u}_1 - 2\lambda \mathbf{u}_1 = 0 \\ S\mathbf{u}_1 &= \lambda_1 \mathbf{u}_1 \end{aligned} \tag{1.36}$$

Therefore, we end up with S$\mathbf{u}_1 = \lambda_1\ \mathbf{u}_1$. Where, λ is a dual variable, and S is a covariance matrix. Meaning that, a matrix S times a vector is a scalar times a vector. This suggests that $\mathbf{u}_1$ and lambda are eigenvector and eigenvalue of S respectively. Generally, eigenvalue (λ) and eigenvector ($\mathbf{u}_1$) of S are the settle points the Lagrangian in equation 1.33.

$$\begin{aligned} \mathbf{u}_1^T S\mathbf{u}_1 &= \mathbf{u}_1^T \lambda \mathbf{u}_1 \\ &= \lambda \mathbf{u}_1^T \mathbf{u}_1 \\ &= \lambda \end{aligned} \tag{1.37}$$

Note that we had the constrain of $\mathbf{u}_1^T\ \mathbf{u}_1 = 1$. This means that if we use any of eigenvector the value/importance of this eigenvector would be the value of the corresponding eigenvalue to that function. Therefore, to maximize the objective function in 1.33, we can choose the enginvector with the maximum eigenvalue. As a result, we sort the eigenvectors according to the eigenvalue, and the eigenvector with the largest eigenvalue will maximize the function at most.

1.3.1 Direct PCA

As the simple trick to compute PCA, in the following section, we will see that the principal components can be computed by computing the covariance matrix, and doing the eigen decomposition on that.

Background: Singular Valued Decomposition (SVD)
You can decompose any matrix $X_{d\times n}$ to three matrix U, Σ and V^T such that $X = U\Sigma V^T$. Here, U is the eigenvectors of X^TX, V is the eigenvectors of XX^T, and Σ is a diagonal matrix with eigenvalues $\lambda_1, \lambda_2, ...$ on the diagonal.

Suppose that the we subtract the matrix of the means $\bar{X}$ from X which is the matrix of the data: $\tilde{X} = X_{d\times n} - \bar{X}_{d\times n}$. Under this formulation, $\tilde{X}\tilde{X}^T$ is the covariance of the matrix $\tilde{X}$. If we do the singular valued decomposition on $\tilde{X} = U\Sigma V^T$. Therefore, columns of U are the eigenvectors or PCs of $\tilde{X}\tilde{X}^T$ when $\tilde{X}\tilde{X}^T$ is the covariance matrix. In this case, Σ will contain the corresponding eignevalues on its diagonal. Now, we can select some subset of eigenvectors in U and compute the projection in the following form:

$$X_{new} = U[:, 1:p]^T X \tag{1.38}$$

We can back to the original space by $\hat{X} = U[:, 1:p]X_{new}$.

Algorithm 3 Direct PCA

1: $\tilde{X} \leftarrow X - \bar{X}$ ▷ $\bar{X}$ is the mean of X
2: U $\leftarrow$ eigenvectors of $\tilde{X}\tilde{X}^T$ ▷ Computing the PCs

---

3: $X_{new} \leftarrow U[:, 1:p]^T X$ ▷ encoding the original data
4: $\hat{X} \leftarrow U[:, 1:p]U[:, 1:p]^T X$ ▷ decoding the X_{new}

5: $x_{new} = U^T x$ ▷ encoding test example x
6: $\hat{x} = U^T x_{new}$ ▷ decoding the x_{new}

1.3.2 Dual PCA

A problem happens with direct PCA when the data dimensional is more than the number of samples.
In SVD, we need to compute the decomposition of matrix $X_{d\times n} = U_{d\times d}\Sigma V^T_{n\times n}$. Here, $U \in \mathcal{R}^{d\times d}$ contains eigenvectors of XX^T, where $X \in \mathcal{R}^{d\times n}$. In some application, the data dimensional is more than the number of samples. For instance, we could have only a few examples of brain images in high dimension per person.

In this case, it is better to decompose $V_{n\times n}$ instead of $U_{d\times d}$ because $V = XX^T$. Since V is orthonormal matrix, we can muliply both sides of the equation by V and obtain:

$$\begin{aligned} X &= U\Sigma V^T \\ XV &= U\Sigma \\ XV\Sigma^{-1} &= U \end{aligned} \tag{1.39}$$

Therefore, we can replace all Us with $XV\Sigma^{-1}$ to solve the discussed problem. This version of PCA is called *dual* PCA. Please note that the reconstruction would be $\hat{X} = XV\Sigma^{-}1\Sigma V^T = XVV^T$.

1.4 Appendix II: Metric Learning

Given a set of data (x, y), where $x \in \mathcal{R}^d$ is a d-dimensional input, and $y \in \mathcal{R}^l$ is the class labels. The goal is to estimate a subspace S such that the subspace such that S would have as much as predictive information about y compared to the original input space. Let $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$, and $Y = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n]$. The aim is to find the transformation matrix U such that $S = U^T X$, and Y will have maximums dependence on U^T at most. [4] To achieve this goal, many different approaches have been proposing such as Fisher's discriminant analysis (FDA), sufficient dimension reduction (SDR) algorithms and metric learning approaches.

Metric learning constructs a Mahalanobis distance over the input space and uses this space to learn a linear transformation from original input space to a Euclidean distance based output space.

Background

Distance metric like Euclidean distance in the form of $|x_i - x_j|^2 = (x_i - x_j)^T(x_i - x_j)$ should satisfy the following four conditions:

[4] The problem is close to manifold learning. The difference is that manifold learning is an unsupervised problem, unlike the metric learning which is a supervised learning.

- $d_{ii} = 0 \quad \forall_i$: distance between $d(\mathbf{x}_i$ and $\mathbf{x}_i) = 0$
- $d_{ij} \geq 0 \quad \forall_{i,j}$
- $d_{ij} + d_{ik} \geq d_{jk} \quad \forall_{i,j,k}$
- $d_{ij} = d_{ji} \quad \forall_{i,j}$

In metric learning, we are trying to learn a new metric by learning a matrix A in the following form:

$$|x_i - x_j|_A^2 = (x_i - x_j)^T A(x_i - x_j) \tag{1.40}$$

which A is any *positive semi-definite matrix*. In other words, $|x_i - x_j|_A^2$ is a distance as long as A is positive semi-definite matrix. Please note that Eq. 1.40 is a type of Maharanis distance. We know any any positive semi-definite matrix can decompose into two matrix $A = UU^T$.

$$(x_i - x_j)^T UU^T(x_i - x_j) = (U^T x_i - U^T x_j)(U^T x_i - U^T x_j) \tag{1.41}$$

This is again a euclidean distance between $U^T x_i$ and $U^T x_j$. Therefore, learning a metric space (transformation matrix A) in the output space is equivalent to the Mahalanobis distance in the input space.

Chapter 2

Questions of Dr. Jean-Francois Lalonde

2.0.1 Describe the mode collapse problem in GANs, and why it occurs.

Answer: The convergence guarantee of the generators in GAN for learning all mods are guaranteed theoretically. In practice, however, reaching the true equilibrium between the generator and discriminator is difficult and not guaranteed Ghosh et al. (2018). Generally, GANs have three main problems: *non-convergence* of the cost function, *vanishing gradient*, and *mode collapse*. Mode collapse happens when the generator **(**G) has biased toward generating limited image diversity. The term *mode* refers to a category or task in this context. In machine learning, almost all of the datasets are multimodal. In MNIST dataset, for instance, there are 10 major modes from digits 0 to 9. Mode collapse happens when the generator of a trained GAN does not produce all of the modes. Please note that GANs are unsupervised learning methods, and the number of modes has not been specified during the training process. However, there is two possibility of mode collapse problem: complete collapse where **G** produces only one mode, and partial collapse where **G** generates only a subset of all modes in the dataset.

An important question is *why mode collapse happens?* Salimans et al. (2016) states that mode collapse occurs when "the generator collapses to a parameter setting where it always emits the same point. When collapse to a single mode is imminent, the gradient of the discriminator may point in similar directions for many similar points." Let us review the idea of GANs. Goodfellow et al. (2014) proposed GAN and used following "minimax" function for training:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\big[\log D(x)\big] + \mathbb{E}_{z\sim p(z)}\big[\log(1-D(G(z)))\big] \tag{2.1}$$

where, $V(.)$ is the objective function over the generator (G) and discriminator (D) functions. The idea of proposing this "minimax" function is to begin an adverserail game between **G** and **D**. The reason which the authors used "minimax" function instead of "maxmin" is that minimization of **G** must be in outer loop to guarantee of covering all data distribution (all modes). In other words, if we reverse the order in Eq. 2.4 from "minimax" to "maxmin", then **G** will map all random samples z to the same output. Therefore, theoretically the optimization algorithm might look perfect. In practice, however, the proposed "minimax" objective function might shows the results of "maxmin" one, which is called mode collapse.

Given a random sample z, the generator **G** generates a fake image $x_{\mathrm{f}} = G(z)$ in order to fool the discriminator **D** such that **D** can not distinguish between x_{f} and a real image x_{r} from dataset. In an extreme case that **G** can be trained without updates to **D**; therefore, we will have only the second expectation in Eq. 2.4. Lets x_{f}^* be the

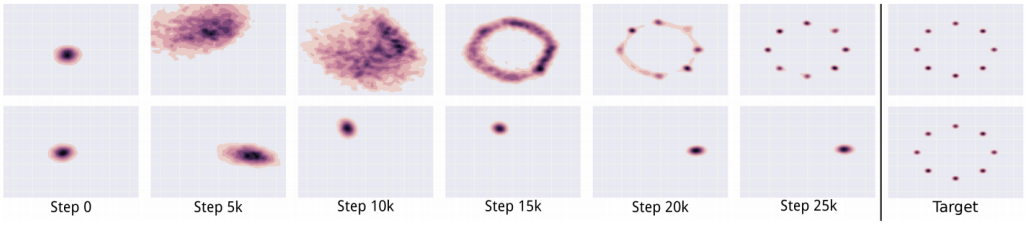


FIGURE 2.1: Mode collapse problem and Unrolled GAN as the solution (Metz et al., 2017). Columns represent the distribution of the generator in the training procedure. The last column shows the target distribution which is a toy 2D mixture of the Gaussian dataset. Top raw shows a proposed unrolled GAN without mode collapse. The bottom row shows the mode collapse problem in standard GAN.

optimal image that fool **D** the most. In this situation, x_f^* will be independent of z,

$$x_f^* = \text{argmax}_x D(x) \tag{2.2}$$

This causes to mode collapses to a single point, because the gradient of the cost function w.r.t z goes twoard zero.

$$\frac{\partial}{\partial z}\mathbb{E}_{z\sim p(z)}\Big[\log(1 - D(G(z)))\Big] \approx= 0 \tag{2.3}$$

Meaning that the objective function does not have any part/constraint to force **G** for generating different modes given z. Therefore, there is a single point on the expectation which **G** thinks that this is the optimal point independent of z. Backing to the training of discriminator (**D**), the generator (**G**) could effectively generator this single point for fooling **D**. Additionally, the gradient of **D** will cause the change of single point the next most valuable mode. In this case, partial mode collapse will happen which produces a subset of modes in the real dataset instead of covering all of the modes in the dataset. Metz et al. (2017) illustrate mode collapse problem by using a toy example of the dataset having 8 modes in 2D. In their experiments, they showed that the GANs with the discussed objective function visit one mode after another shown in Fig 2.1.

2.0.2 From the computer vision literature, present 3 different approaches that provide more robustness to the model collapse problem.

Answer: In other to address the mode of collapse problem, two different research paths are followed. In the first path, several studies like Unrolled GAN Metz et al. (2017) try to reach a better optimum. In the second research direction such as MAD-GAN of Ghosh et al. (2018), the goal is to capture the diverse modes by explicitly enforcing GANs. In the following section, three examples of theses algorithms are covered.

Study I): Multi-Agent Diverse Generative Adversarial Networks (MAD-GAN)
Inspired by multi-agent algorithm (Abadi and Andersen, 2016) and coupled GAN (Liu and Tuzel, 2016), in one of the latest studies related to mode collapse, Ghosh et al. (2018) proposed Multi-Agent Diverse GAN (MAD-GAN) and its condition version to addresses the discussed mode collapse problem. Unlike classic GAN which contains one generator and one discriminator, the idea of MAD-GAN was based on

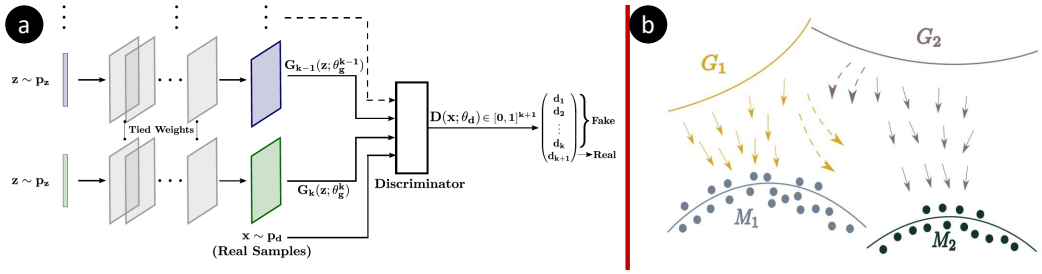


FIGURE 2.2: Multi-Agent Diverse Generative Adversarial Networks (MAD-GAN) (Ghosh et al., 2018). (a) schematic view of MAD-GAN.(b) visualization of various generators ($\mathbf{G}_1$) for capturing different clusters of modes (M_1, M_2).

using the k number of generator network and one discriminator network. The purpose of multiple generators is to enforce the different generator to capture all of the modes. Intuitively, the author argues that the discriminator will learn to push the various generators towards different modes.

The idea of MAD-GAN is shown in Fig. 2.2 (a). Here, the aim of the i^{th} generator is to maximize the mistakes of the common discriminator. In order to reduce the number of redundant computations and fast convergence, all k number of generators are tying the most initial layer parameters in the case of homogeneous data (like images of face or birds). In other words, they share information for the images belong to the same class like face or birds. However, this tying is not done in the case of diverse-class datasets. Given a random sample z, the i^{th} generator minimizes the following objective function:

$$\mathbb{E}_{x\sim p_{data}}\left[\log D_{k+1}(x;\theta_{data})\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1 - D_{k+1}(G_i(z;\theta_g^i);\theta_{data}))\right] \tag{2.4}$$

For parameter update using gradient descent, the following object function is computed as:

$$\nabla_{\theta_g^i}\ \log(1 - D_{k+1}(G_i(z;\theta_g^i);\theta_{data})) \tag{2.5}$$

For the discriminator, the gradient would be: $\nabla_{\theta_{data}} \log D_j(x;\theta_{data})$. Where, $D_j(x;\theta_{data})$ is the j^{th} index of $D(x;\theta_{data})$. Figure 2.2 (b) presents the visualization of different generators ($\mathbf{G}_1$, $\mathbf{G}_2$) for different modes.

Study II): UNROLLED GENERATIVE ADVERSARIAL NETWORKS

Answer: As we discussed so far, the generator and discriminator are playing an adversarial game in GAN. Intuitively, in a game, we can give several tries to our opponent before taking our turn. This is the idea of unrolled GAN which generator unroll to k steps on how the discriminator may optimize itself. This discourages the generator for the local optimal exploit and decreases the chance of overfitting the generator. In practice, unrolled GAN is one of the well-known ways of addressing the mode collapse problem.

Metz et al. (2017) adapted the generator's optimization function w.r.t an unrolled

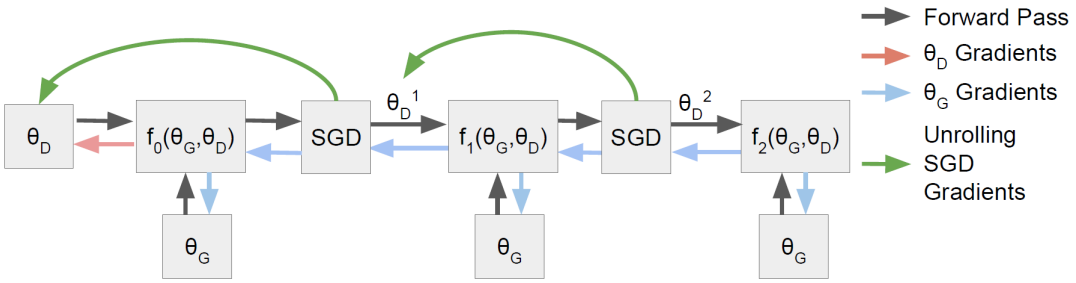


FIGURE 2.3: An illustration of unrolled GAN Metz et al. (2017) with 3 unrolling steps.

optimization of the discriminator. In learning procedure, GAN has difficulty of finding optimal parameters θ_G^* in the following "minimax" function:

$$\begin{aligned} \theta_G^* &= \arg\min_{\theta_G} \max_{\theta_D} f(\theta_G, \theta_D) \\ &= \arg\min_{\theta_G} f(\theta_G, \theta_D^*(\theta_G)) \\ \theta_D^*(\theta_G) &= \arg\max_{\theta_D} f(\theta_G, \theta_D) \end{aligned} \tag{2.6}$$

where $f(.)$ have the same form of $V(.)$ in the Equation 2.4. Here, the local optimum of the parameters of the discriminator θ_D^* can be expressed as an iterative optimization process:

$$\begin{aligned} \theta_D^{k+1} &= \theta_D^k + \eta^k \frac{df(\theta_G, \theta_D^k)}{d\theta_D^k} \\ \theta_D^*(\theta_G) &= \lim_{k \to \infty} \theta_D^k \end{aligned} \tag{2.7}$$

where η^k is the learning rate. However, in unrolled GAN, the author create a new update function by unrolling K steps: $f_k(\theta_G, \theta_D) = f(\theta_D, \theta_D^K(\theta_G, \theta_D))$. In this case, $K = 0$ corresponds to original GAN, $K = \infty$ corresponds to the optimum generator objective function $f(\theta_G, \theta_D^*(G))$. Figure 2.3 represents an unrolled GAN with 3 unrolling. In this example the model backpropagates the generator gradient (blue arrows). The update of discriminator does not depend on the unrolled optimization (red arrow). The green arrows represent one of each steps in k unrolled GAN which uses the gradients of f_k w.r.t. θ_D^k (see equation 2.7).

Study III): Dist-GAN: An Improved GAN using Distance Constraints
Answer: To alleviate mode collapse and optimization problem of GAN, Tran, Bui, and Cheung (2018) proposed the generator with two distance constraints using an Autoencoder to improve the generator.

- *latent-data distance constraint*: enforces compatibility between the distance of latent sample and corresponding data sample.
- *discriminator-score distance constraint*: for aligning the distribution of real samples with the distribution of the generated samples.

Before designing these distance constraints, the author, first, visualize the latent sample in 2D space between -1 and 1. Please see the visualization of mode collapse in 2.4 (a) where many samples from a large region of latent space are collapsed into

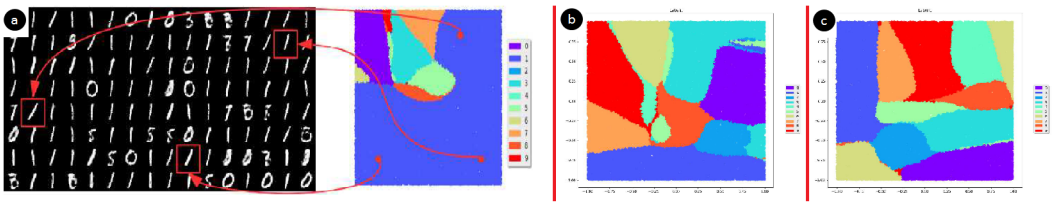


FIGURE 2.4: (a) mdoe collapse, (b) Dist-GAN (without latent-data distance) and (c) Dist-GAN (with the proposed latent-data distance)

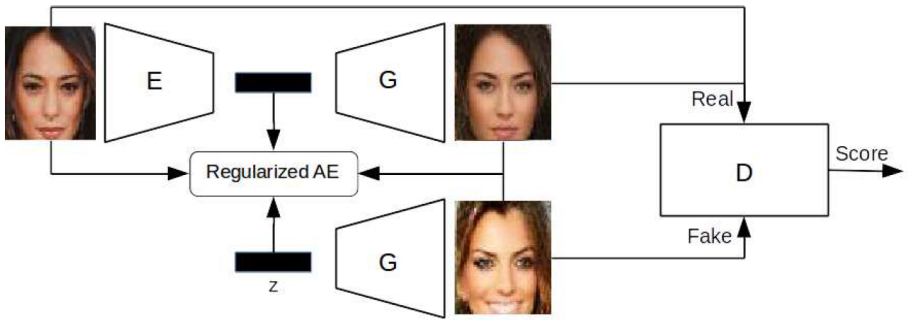


FIGURE 2.5: The architecture of Dist-GAN with Encoder (E), Generator (G) and Discriminator (D)

the same digit. Intuitively, they argue that a generator G_θ has mode collapse many latent samples mapped to small regions of the data space:

$$x_i = G_\theta(z_i), x_j = G_\theta(z_j); d_{data}(x_i, x_j) < \delta_x \tag{2.8}$$

where, x_i is the corresponding synthesized sample by G_θ which its latent sample is z_i. $d_{data}(.)$ is some distance metric in the data space, and δ_x is a small threshold in the data space. Based on this, a distance metric (*distance constraint*) $d_{latent}(.)$ in the latent space is proposed in this study to address mode collapse in the following form:

$$d_{latent}(z_i, z_j) > \delta_z \rightarrow d_{data}(x_i, x_j) > \delta_x \tag{2.9}$$

where δ_x, and δ_x are thresholds. The purpose of $d_{latent}(.)$ is to restrain the generator. To apply this idea, an Autoencoder (AE) is used is used to encode data sample into latent variables, and used these variable to direct the generator's mapping. Here, the generator is the decoder of AE. Additionally, in order to regularize the generator and preventing from being the collapse, the following *latent-data distance constraint* is used in the following form

$$\begin{aligned} \min_{w,\theta} L_R(w,\theta) &= \lambda_r L_W(w,\theta) \\ \min_{w,\theta} ||x - G_\theta(E_w(x))||_2^2 &= \lambda_r L_W(w,\theta) \end{aligned} \tag{2.10}$$

where, w, θ are the parameters of the encoder and generator respectively. Please note that $L_R(w,\theta)$ is the objective function of conventional AE. Figure 2.5 presents the architecture of Dist-GAN. For the "real" image, the reconstructed samples by AE are considered. As shown in the figure, a input sample x_i, the latent of that sample z_i, the reconstructions $\hat{x}$ of AE, and the generated image of gan all used to regulate the AE. The training of the encoder E_w, generator G_θ, and discriminator D_γ is done

in the following way:

- Fix D_γ and train E_w and G_θ to minimize 2.10.
- Fix E_w, G_θ, and train D_γ to minimize

$$\begin{aligned}\min_\gamma \mathcal{L}_D(w,\theta,\gamma) = &-(\mathbb{E}_x \log\sigma(D_\gamma(x)) + \mathbb{E}_z \log(1-\sigma(D_\gamma)(G_\theta(z)))) \\ &+ \mathbb{E}_x \log\sigma(D_\gamma(G_\theta(E_w(x)))) - \lambda_p \mathbb{E}_{\hat{x}}(L_p)\end{aligned} \tag{2.11}$$

  Here, σ denotes the sigmoid function and $\mathbb{E}$ denotes the expectation. $L_p = ||\nabla_{\hat{x}} D_\gamma(\hat{x})||_2^2 - 1)^2$ is the gradient penalty for the discriminator objective, and λ_p is a penalty coefficient.

- Fix E_w, D_γ and train G_θ to minimize:

$$\min_\theta \mathcal{L}(\theta) = |\mathbb{E}_x \sigma(D_\gamma(x)) - \mathbb{E}_z \sigma(D_\gamma(G_\theta(z)))| \tag{2.12}$$

  Here, l_1 distance is used to align the synthesized sample distribution to real sample distribution.

2.1 Question 2: Object detection

2.1.1 Present a detailed description of the 3 different meta-architectures that are Faster R-CNN, R-FCN, and SSD.

Huang et al. (2017) analyzed Faster R-CNN, R-FCN, and SSD convolutional based object detection algorithms in terms of speed, memory usage and accuracy. In this question, a detailed description of these algorithms are covered in different subsections.

Faster Region Based Convolutional Neural Network (Faster-RCNN)
Faster R-CNN (Girshick, 2015) is composed of two modules: a deep fully convolutional network and a detector. While the first module, a deep fully convolutional network, proposes the regions of interest by another network called Region Proposal Network (RPN), the second module which is called Fast R-CNN uses these RoIs for object detection. In other words, Faster R-CNN is region RPN plus Fast R-CNN of Girshick (2015). Before discussing RPN, let discuss what is Fast R-CNN, and *why do we need RPN*.

In Fast R-CNN, the fast version of region-based CNN (R-CNN) algorithm, the whole input image is given to a deep convolutional network with several conv layers and max-pooling layers to produce a conv feature map. Like R-CNN, a selective search algorithm is used in Fast R-CNN to extract region proposals. However, unlike R-CNN, we project these region proposals on the feature map in Fast R-CNN (instead of cropping these region proposals from the input image). This decrease the amount of convolutional computation across the entire image when we have many many region proposals. Fast R-CNN contains a region of interest (ROI) pooling layer. This layer extracts a fixed-length feature vector from the different sized feature maps. Then, each fixed-length feature vector is fed into a sequence of FC layers which is branched into two parts: one that produces softmax probability, the

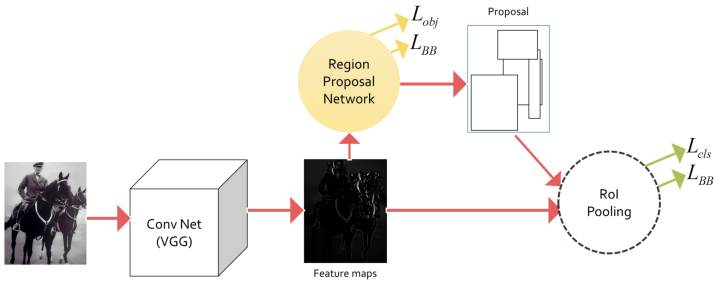


FIGURE 2.6: Schematic overview of R-FCN.

other is the 4 real value (coordinates of the bounding box around the object) for each of K object classes. A RoI is a rectangular window in a conv feature map, and RoI pooling layer uses max-pooling in order to convert the feature inside any valid RoI into a small feature map. This small feature map has a fixed spatial size. RoI pooling layer contains the following two steps:

- Dividing the RoI window into the grid (the fixed spatial extent) of sub-windows.
- Performing a max pooling over the sub-windows in the corresponding grid.

Please note that the RoI pooling layer is applied independently to each feature map channels, as in standard max pooling. In fact, the idea of Fast R-CNN is to pass the input image through some convolutional layers all at once rather than processing each region separately. After the fully connected layer, the classification is done by a softmax layer, and there is a linear model for bounding-box regressions. In the training phase, both of the classification and regression losses. In fact, the computational time of Fast R-CNN is less than a second. However, the computing region proposal algorithm takes time. For example, the used selective search algorithm takes about 2 seconds. Therefore, the overall systems take 2.3 seconds per frame, and Fast R-CNN is limited by computing of these region proposals in a real-time application.

To overcome the discussed problem, Region Proposal Network (RPN) is proposed in Faster R-CNN (Girshick, 2015) instead of selective search algorithms. RPN a network which learning to propose regions given an image. Figure 2.6 represents the idea, the first part of the network for both of RPN and Fast R-CNN assume a common set of convolutional layers. For example, VGG-16 Simonyan and Zisserman, 2014 can be used to extract feature maps for both RPN and Fast R-CNN. The schematic overview of Faster R-CNN is shown in Figure 2.6. In RPN, the authors slide k proposed anchor boxes over the elements of the input feature map which is shown in 2.7. They used 9 different anchor in different size and scales. This network is trained using two different losses: classification loss L_{obj} (specifies if there is an object in the proposed region) and L_{BB} (regression error between the ground truth bounding box and the proposed region).

After proposing regions with RPN, the rest of the network is Fast R-CNN which detect objects by using two losses. The first loss is classification loss L_{cls}, and the second one is regression loss of bounding box L_{BB}. Faster R-CNN with RPN decreases the object detection time of image from 2.3 second per-frame (of Fast R-CNN) to 0.2 of a second.

Region Based Fully-Convolutional Network (R-FCN)
As discussed above, Faster R-CNN is based on a region proposal network (RPN), and is composed of two stages. First, RPN estimate region proposal. Then, RoI

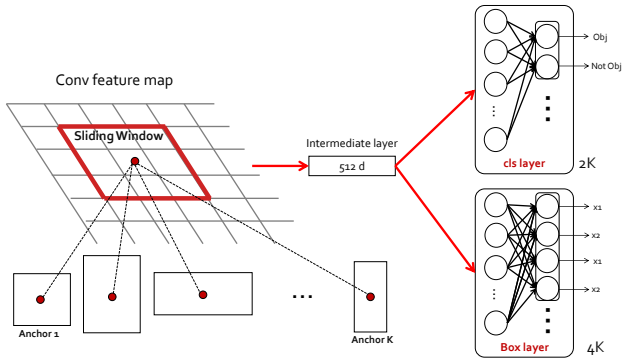


FIGURE 2.7: Schematic overview of region proposal network (RPN).

pooling layer is ended by fully connected (FC) layers for bounding box regression and classification. In Fast R-CNN, each RoI has its own FC layer which increases the number of parameters and time complexity. To decrease these high number of parameters, Dai et al. (2016) proposed a new region based algorithm.

Region-based fully-convolutional network (R-FCN) (Dai et al., 2016) is a RPN based method without having FC layer after RoI pooling. R-FCN is considered as one of the meta-architectures in CNN-based object detection methods. In R-FCN, the author moves all of the time-consuming parts (at the end of Faster R-CNN) to before RoI pooling layer. For classification purpose, R-FCN uses average voting over all of the region proposals by using the same set of the score. As a result, the computation cost of R-FCN is lower than Faster R-CNN using same settings. Because there is no learnable layer after RoI layer.

In R-FCN, first, RPN extracts the candidate regions. Second, they shared the features between RPN and R-FCN. The aim of R-FCN is to classify the RoI (proposed by RPN), where all of the learnable weights are computed on the whole image. The proposed system also contains a *position-sensitive score maps* which are proposed to solve the dilemma between translation invariant in image classification and translation invariant in the object detection.

As Figure 2.8, the last layer of convolutional network (before positive sensitive score maps) contains $k^2(C+1)$ convolutions. Meaning that we have k^2 feature map {top-left (TL), top-center (TC), ... , bottom right (BR)} for each class of C classes. Please note that there is one additional ($+1$) class for detecting background (no object). After this final layer of CNN, R-FCN contains a position-sensitive RoI pooling layer which aggregates the outputs of the discussed convolutional layer. In this pooling layer, the convolutional layer is pooled with the area and the same color Figure 2.8. Here, an average voting is performed to generate a vector of length $(C+1)$. Finally, a softmax layer performed on this vector. To find the bounding box around the object, R-FCN uses class-agnostic bounding box regression, which the regression is shared among the classes. Here, a sibling $4k \times k - d$ convolutional layer is attached to the discussed $k^2(C+1)$ convolutional layer to detect the bounding box.

For feature extraction purpose before the RoI, 100 conv layers of ResNet-101 of (He et al., 2016) (pretrained from ImageNet) is used before positive-sensitive score maps.

Single Shot Multibox Detector (SSD)

Region proposals network (used in Faster R-CNN and R-FCN) makes the discussed method to be slow for real-time object detection application. Liu et al. (2016) proposed a popular algorithm for real-time object detection which is called single shot

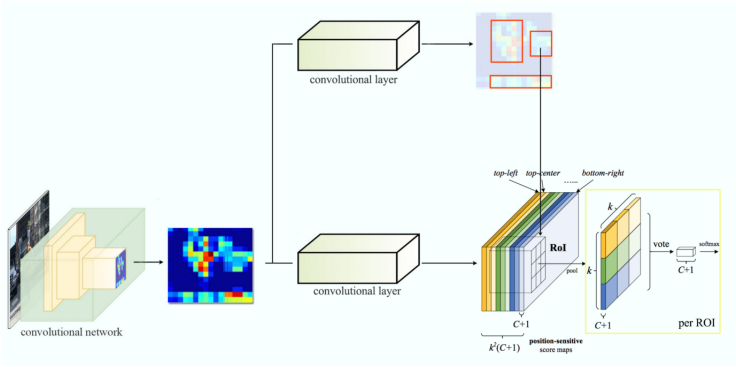
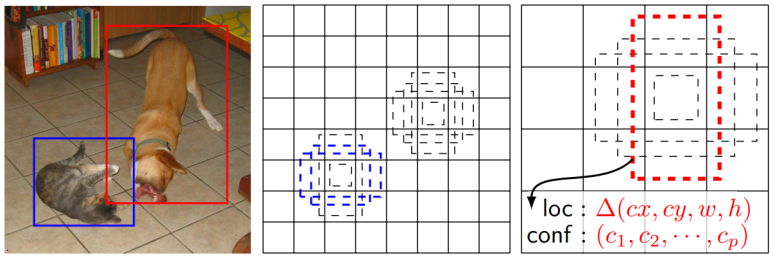


FIGURE 2.8: Schematic overview of R-FCN. The figure is adapted version of the figure in (Dai et al., 2016). This figure is taken from the this link.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

FIGURE 2.9: SSD framework.

multi-box detector (SSD). Intuitively, the idea of SSD is based on the receptive field in deep CNN architectures. In recently proposed architectures such as VGG (Simonyan and Zisserman, 2014) and ResNet (He et al., 2016), the receptive fields coverage gets larger while going deep and deep in the networks. As a result, more abstract representations are obtained in the deep layers compared to the shallow layers. Based on this intuition, SSD aims to predict small sized objects in a shallow network because small objects don't need large receptive fields. As Fig. 2.9 shows, for example, the figure maps of size 8×8 and 4×4 can capture different object size in an input image.

In SSD, we first we discretize the outer region of bounding boxes by a set of default boxes with different ratios and scales per feature map location. Then, the network outputs one score per class for each of the boxes. Next, the network adjusts the bounding box for a better match with the shape of the object.

As discussed above, SSD uses multiple independent layers for object detection. This method uses lower resolution features in deeper layers of CNN for detecting larger objects. For instance, feature maps with the size of 4×4 are used for detection of larger objects. Figure 2.10 illustrates the idea of SSD with VGG16. As the figure shows, Liu et al. (2016) add six extra convolution layers for object detection. Each of these auxiliary feature layers uses a set of convolutional filters to produce a set of detection predictions. In each of the feature layers of size $m \times n \times p$ (p is the size of the channel), they used the small kernel of size $3 \times 3 \times p$ for object detection.

SSD consists of two parts which eliminate region proposals requirements, and it speeds up the object detection procedure. Let see how it works. First, Liu et al. (2016) uses **Conv**4_3 of VGG16 (Simonyan and Zisserman, 2014) architecture to extract feature maps which are used to detect the objects. In this layer, the feature map has the size of 38×38. SSD makes 4 object prediction per each cell. Therefore,

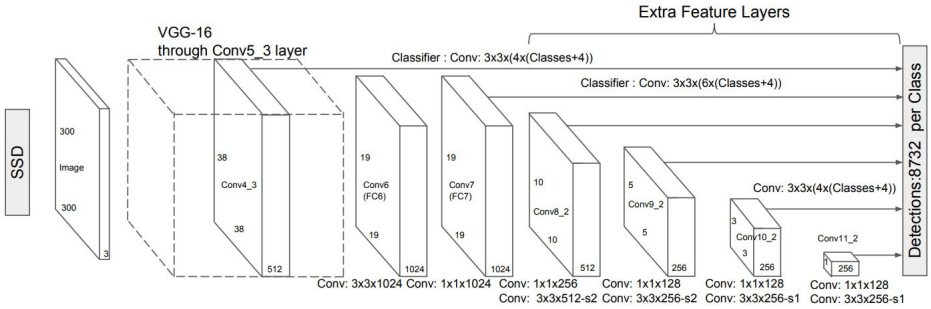


FIGURE 2.10: "SSD model adds several feature layers to the en of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences." Liu et al., 2016

we will have $5776 = 38 \times 38 \times 4$ predictions in total. Second, the authors used 3×3 convolution filters for each cell to detect the objects. Each filter gives us one score per class, one additional score for no object, and the location of the bounding box. SSD uses the single neural network and makes multiple predictions (boundary box, and confidence score) which is called multi-box.

2.1.2 Highlight the similarities and differences of the meta-architectures.

In the previous question, we discussed three well-known object detection ideas: Faster R-CNN, R-FCN, and SSD. In this question, we will review the differences and similarities between these methods. Most of the materials in this question are based on an outstanding paper by Huang et al. (2017) which provides an analysis of speed, memory usage and accuracy of the three methods. In this study, the author views Faster R-CNN, R-FCN and SSD as "meta-architectures".

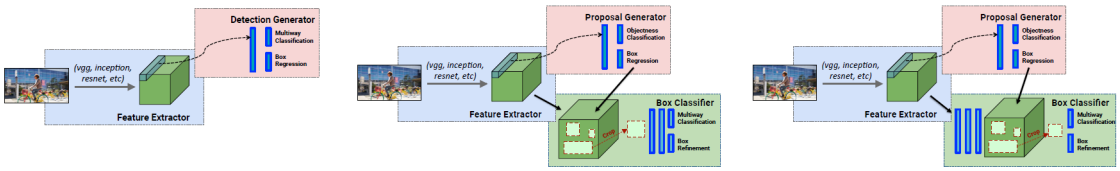
Similarities

The similarities between Faster R-CNN, R-FCN and SSD are as follow:

- These methods are single-model/single pass detector. None of the meta-architectures are based on ensambling and multi-crop methods.
- These methods contain single CNN for feature extractor.
- These methods are trained by a mixed regression and classification objectives.
- These methods use the idea of the sliding window for prediction.
- Both of Faster R-CNN and R-FCN:
  - use the idea of region proposal network (RPN)
  - use the anchors methodology (please see the previous question)

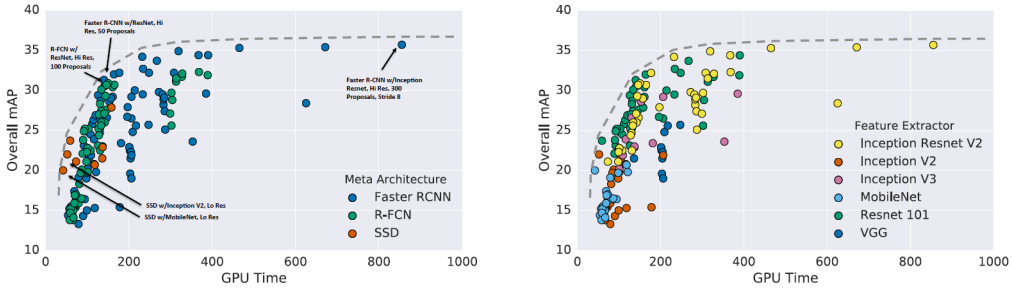
Differences

The overall high level view of the meta-architectures (Faster R-CNN, R-FCN, and SSD) are showin in Figure 2.11. The main difference between these methods are as follow:



(a) SSD (b) Faster R-CNN (c) R-FCN

FIGURE 2.11: High level schematic over view of SSD, Faster R-CNN, and R-FCN

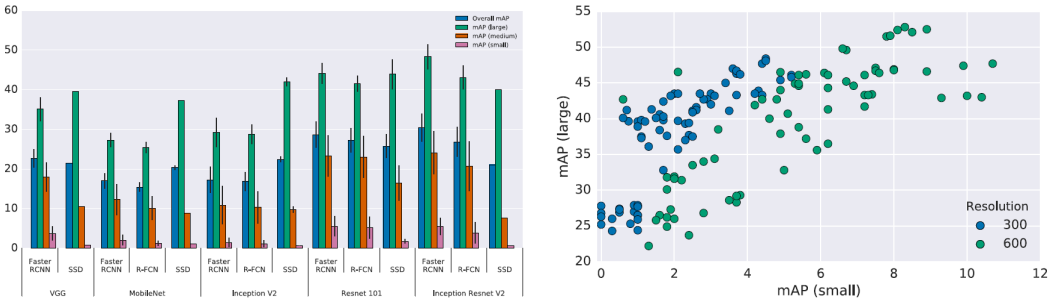


(a) (b)

FIGURE 2.12: mean average precision (mAP) vs gpu wallclock time colored by (a) meta-architecture and (b) feature extractor.

- Unlike region based meta-architectures (Faster RCNN and R-FCN), SSD does not contain region proposal network (RPN). This makes SSD fast compared to Faster R-CNN and R-FCN. For more detail please see the previous question.
- Unlike Faster R-CNN, the box classifier module of R-FCN crops are taken from the last layer of features (instead of cropping features from the right same layer where region proposal are predicted). This leads to a significant reduction in the number of parameters in R-FCN compared to Faster R-CNN.
- Unlike Faster R-CNN which is based on the RoI mechanism, R-FCN contains position-sensitive cropping mechanism instead of RoI. The creators of R-FCN argue that position-sensitive cropping is invariant to translation Dai et al. (2016).
- Figure 2.12 represents overall mAP versus time in different meta-architectures (a) and feature extractors (b). According to this figure Faster R-CNN tends to be slower than R-FCN and SSD, but the accuracy of Faster R-CNN is more than the others.
- The best balance between speed and accuracy seems to be R-FCN model with Residual Networks as the feature extractor. However, by a little accuracy scarification, it has been shown that Faster R-CNN can speeds up by decreasing the number of region proposal to 50, and it can reach to speed/accuracy balance of R-FCN.
- SSD model with Inception v2 and Mobilenet are the more accurate of the fastest models.

2.1.3 Discuss which of the three meta-architectures would be more appropriate for the work you are proposing to do.



(a) (b)

FIGURE 2.13: (a) mAP for each object size by meta-architecture and feature extractor (for size 300 inputs). (b) mAP on small objects vs mAP on large objects colored by input resolution.

Answer: In our proposal, we are discussing one possible research path could be extending one of the existing meta-architecture meta-learning frameworks. In other words, we are proposing to adopt one of the meta-architectures like Faster R-CNN to distance based few shot learning. The other possibility could be working on visual place recognition in few-shot learning setting. Huang et al. (2017) guide the reader to select the best object detection architecture (Faster R-CNN, R-FCN, and SSD) based on memory usage, the speed of the algorithms, and accuracy's of the algorithms. Therefore, as a purpose of this question, it would be very efficient to first have a review the comparison of these methods before using them (Thanks!).

In self-driving car and robotics, one of the big challenges is visual place recognition. Therefore, if we choose to work on visual place recognition topic, real-time performance would be very important for us. Additionally, we must take care of the accuracy, because k shot learning setting might need robust accuracy of the used meta-architecture.

Figure 2.13 illustrate the accuracy compression between Faster R-CNN, R-FCN, and SSD in different CNN feature extractors. As the subplot (a) in the figure shows, Faster R-CNN is almost better than R-FCN and SSD in terms of accuracy. As a result, Faster R-CNN could be a good choice for our project. However, we should consider the real-time performance of the meta-architecture. It is shown that Faster R-CNN is slower than R-FCN and SSD in terms of speed. Huang et al. (2017) showed that Faster R-CNN can get faster by decreasing the number of anchors, but they also argue that this might cause to decrease in the accuracy of the object detector. However, the location/places are varying slowly compared to other temporal scenes like movies. Additionally, we are considering to work in within city place recognition. Therefore, the speed of Faster R-CNN which is 0.2 second per frame is not a big issue in our case. Considering all of these factors, I think currently Faster R-CNN could be a good option for real-time visual place recognition.

Chapter 3

Questions of Dr. Denis Laurendeau

3.1 Question 1. Object recognition Part 1

Question: A classical approach for object recognition or object retrieval in image databases is to use invariant descriptors to characterize object features in the images. Ideally, the descriptors should be invariant to scale, viewpoint and scene illumination. Several descriptors have been proposed in the literature including SIFT, SURF and ORB descriptors. In this question, describe the main features of these three descriptors in terms of invariance, computational load, the potential of a real-time application, and recognition performance.

Answer: In the following section, we cover the the a brief presentation of the SIFT, SURF and ORB. To make a consistent comparison for computational load and recognition performance, we will refer a review study of these three methods by Karami, Prasad, and Shehata (2017).

I) *SIFT*
Scale Invariant Feature Transform (SIFT) proposed by (Lowe, 2004) with the aim of solving intensity and viewpoint change in matching features, rotation and affine transformation. In this study, the author proposed following steps:

1. Estimate a scale space extreme using Difference of Gaussian (DoG)
2. Localization of key point in the key point candidates by elimination of the low contrast points
3. Assigning a key point orientation based on the gradient of the local image
4. Generating a descriptor for computing the local image descriptor

SIFT is very successful in terms of invariance, and it is scale invariance. It is also robustly invariant to image rotations. Additionally, SIFT is invariant to the limited affine variations. In terms of computational load, SIFT has high computational cost (main drawback) (Lowe, 2004), Karami, Prasad, and Shehata (2017). SIFT's performance in a test of over 24 NTSC resolution images in PASCAL dataset is 5228.7ms time per frame (ms) (Krig, 2016). Therefore, in most of real-time application SIFT's speed could be insufficient. This is also shown in (Lowe, 2004), Karami, Prasad, and Shehata (2017). Dispute the slowness, in terms of recognition performance, SIFT is showing better recognition performance Karami, Prasad, and Shehata (2017).

II) *SURF*
Speeded Up Robust Features (SURF) (Bay et al., 2008) is also based on analysis of

images in Gaussian scale space. Using box filters, SURF approximate the Difference of Gaussian (DoG). SURF detector exploits integral images using determinant of Hessian Matrix to improve feature detection speed. In other words, SURF finds the points of interest based on BLOB detector which uses Hessian matrix. Feature matching using SURF allows faster matching.

SURF features are invariant to rotation, to scale, but they have little affine invariance. In terms of computation, SURF is faster than SIFT. The speed performance of SURF in a test of over 24 NTSC resolution images in PASCAL dataset is 217.3ms time per frame (ms) (Krig, 2016). It is much faster that SIFT (with 5228.7ms), and it could be use in some of the real-time application. However, the recognition accuracy of SURF is lower than SIFT in some of the cases (Karami, Prasad, and Shehata, 2017).

III) *ORB*
Oriented FAST and Rotated BRIEF (ORB) (Rublee et al., 2011) algorithm is an adapted version of Features from Accelerated Segment Test (FAST) algorithm (Rosten and Drummond, 2006) and BinaryRobust Independent Elementary Features (BRIEF) (Calonder et al., 2010). Initially, FAST is used to determine the key points. BRIEF descriptors are unstable in the case of rotation; therefore, a modified version of this algorithm is used.

ORB features are invariant to scale, to the rotation, but ORB features are limited affine changes. In terms of computation, ORB is faster than the discussed meta-architectures. ORB is much faster than SIFT (with 5228.7ms) and SURF (217.3ms), and it is better in terms of real-time (Krig, 2016). ORB showed an outstanding performance in a test of over 24 NTSC resolution images in PASCAL dataset is 15.3ms time per frame (ms). Rublee et al. (2011) argue that ORB performing as well as SIFT in many cases, but Karami, Prasad, and Shehata (2017) presents some accuracy decrease in ORB compared to SIFT. In (Krig, 2016), the robustness of the method is summarized as follow:

- SIFT: brightness, contrast, rotation, scale, affine transforms, noise
- SURF: scale, rotation, illumination, noise
- ORB: brightness, contrast, rotation, limited scale

3.2 Question 2. Object recognition Part 2

Question: A very popular approach for object description is the Bag-of-Words approach. Explain the main properties of the Bag-of-Words approach for image recognition.
Answer: Bag-of-Words (BOW) is a very simple, intuitive, and one of the effective and popular approaches which are used in computer vision and NLP. Under computer vision, BOW can be used for recognition. In NLP, we count the number of each word in the input document, and the frequency of each word determines the keywords of the document by making a histogram of the words. Therefore, the input document is supposed to be a BOW. In computer vision, the idea is the same. In BOW for image classification, the input is image features instead of words in NLP.

In computer vision, the image features are supposed to be a unique pattern which can find in an image. These image features are considered to be the key points and description. These key points are invariant to rotate, shrink, or scale. We use

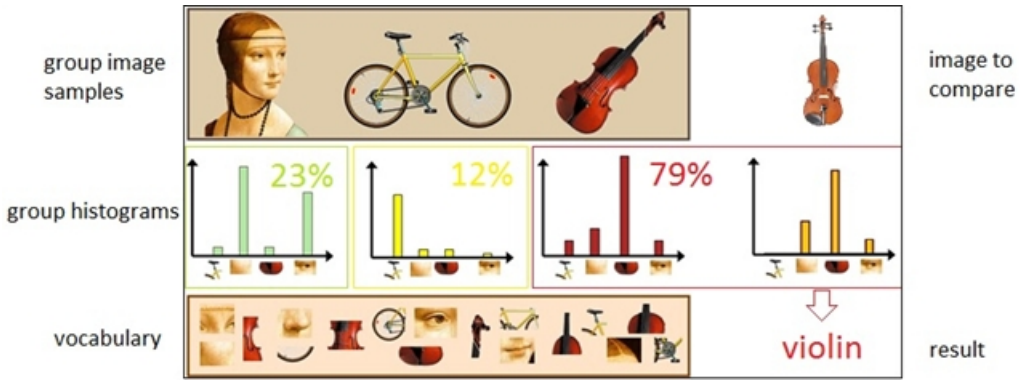


FIGURE 3.1: Bag-of-Word for image recognition. Note, this figure is taken from the this link.

the histogram of these key points for comparing them with the histogram of labeled key points. BOW for image recognition is as follow:

1. Build a visual word vocabulary with a clustering algorithm like k-means algorithm as figure 3.2 left-bottom-row shows. Here, the number of classes (the k of k-means) is equal to the number of visual words in the vocabulary.
2. Store the vocabulary in a file.
3. Compute the normalized histogram (group histograms). Here, median histogram or average histogram can be used shown in figure 3.2 left-middle-row.
4. Compute the histogram of the input image right-middle-row in figure 3.2.
5. Compare the histogram of the input with the histogram of the group.
6. Classify the image as belonging to the category of the most similar group right-bottom-row in figure 3.2.

3.3 Question 3. 3D object descriptors

Question: In 3D-vision, a very popular image descriptor is the spin-image descriptor. Explain the main properties of spin image descriptors. Are these descriptors invariant to scaling?
Answer:
The spin image (proposed by Johnson (1997)) is used for object recognition in 3-D and surface matching. Spin image technique provides a surface representation of an object in an object-oriented system. Unlike viewer-oriented coordinate system, the object-oriented system does not base on the surface viewpoint of the observer. Instead, the object-oriented system (used in the spin image) is based on fixed coordinate systems with respect an object or surface. The experimental results show that spin images are effective even with cluttered and occlusion.

The original version of spin images (Johnson, 1997) is not scale invariant. Darom and Keller (2012) present a new type of scale-invariant spin image local descriptor which is a scale-invariant version of the spin image descriptor. The local description is invariant to rigid transformations. This is because the encoded image has coordinates of points on the surface of an object. Additionally, spin images are translation

invariant and rotation invariant. Johnson (1997) states that spin images are useful descriptors because of:

- pose invariant / object-centered
- simple to compute / transformation of the data
- scalable from local to the global representation
- robust to clutter and occlusion
- impose minimal restrictions on the surface shape
- widely applicable to problems in 3-D computer vision

In the spin image, a polygonal mesh with N number vertices represent the oriented points (O). As an important element of the spin image, oriented points are 3D surface points with associated direction. Using these oriented points, a 2-D map called **spin map** (S_O) are generated. Intuitively, the spin map is the projection of surface points x on the 2-D coordinate (α, β). This is done by following projection function:

$$S_O(x) \rightarrow (\alpha, \beta) = \left(\sqrt{||x-p||^2 - (n.(x-p)^2},\ \ n.(x-p)\right) \tag{3.1}$$

where, (n, p) is the 2-D basis correspond tot he oriented point O. We need one oriented point in order to create. After creating spin map image, Bilinear interpolation is used to generate the 2-D array. Bin size and the size of an expressed object in oriented point coordinates determines the size of the spin image.

In object recognition, first, we generate spin images of all surface points and store them by representing each model in the form of a polygonal mesh. Then, we select an oriented point (randomly) that is also represented in the mesh form. Next, we generate the spin image of the oriented point to correlated it will all of the stored spin images in the first stage.

In some of the application, the spin image can be slow. In matching, for example, principal component analysis (PCA) is used by Murase and Nayar (1995) be used to compression and speeding up the process.

3.4 Question 4. Object recognition Part 3

Question: Geometric hashing has been proposed several years ago as a simple and efficient real-time approach for object recognition. Explain the two principal steps (offline training and online recognition) composing the geometric hashing approach for object recognition. Can this approach be extended to 3D objects?

Answer: In computer vision, geometric hashing (GH), a highly efficient technique, is developed for matching some geometric features with a database of such features. In this technique, matching is possible in the case of databases transformations or in the absents of some information in the dataset (Wolfson and Rigoutsos, 1997). For the first time, the method is a proposed by (Lamdan and Wolfson, 1988), and its application in both 2-D and 3-D presented. Specifically, the authors illustrate the recognition of 3-D objects from 2-D gray images in occluded scenes.

GH is a model-based object recognition system which addresses both *object representation* and *matching*. In this method, an object is represented as a set of geometric features like points and lines. Additionally, the geometric relations of the features are encoded by a minimum set of features. GH is consists of two stages: offline and online. The schematic overview of this algorithm is shown in Figure 3.2. Let discuss each of the stages in detail. The offline stage (which is also called modeling) is done in the following steps:

1. Extract a set of features from a model image
2. Transform all of the features to a new coordinate system for each feature pair (x_1, x_2) or basis. In the new coordinate system, a new coordinate system one axes goes from x_1 to x_2.
3. Store all feature coordinates in a *Has Table*, a quantized histogram Record (model, basis).

In the online or recognition stage,

1. Extract features from the given scene or target image
2. Transform all features of the target image using a chosen basis transformation done on the model image during the online stage. Transformation should be done such that each transformed coordinate pair casts a vote for the bases which were accumulated in the hash table bin they fall into.
3. Recognize the object into the target image if a basis gets a number of votes higher than a pre-defined threshold, and
   - Find the best-induced transformation
   - Check the edges of the object against the scene edges. If the check is passed, the recognize the object in the scene.
4. Otherwise, go to (2)

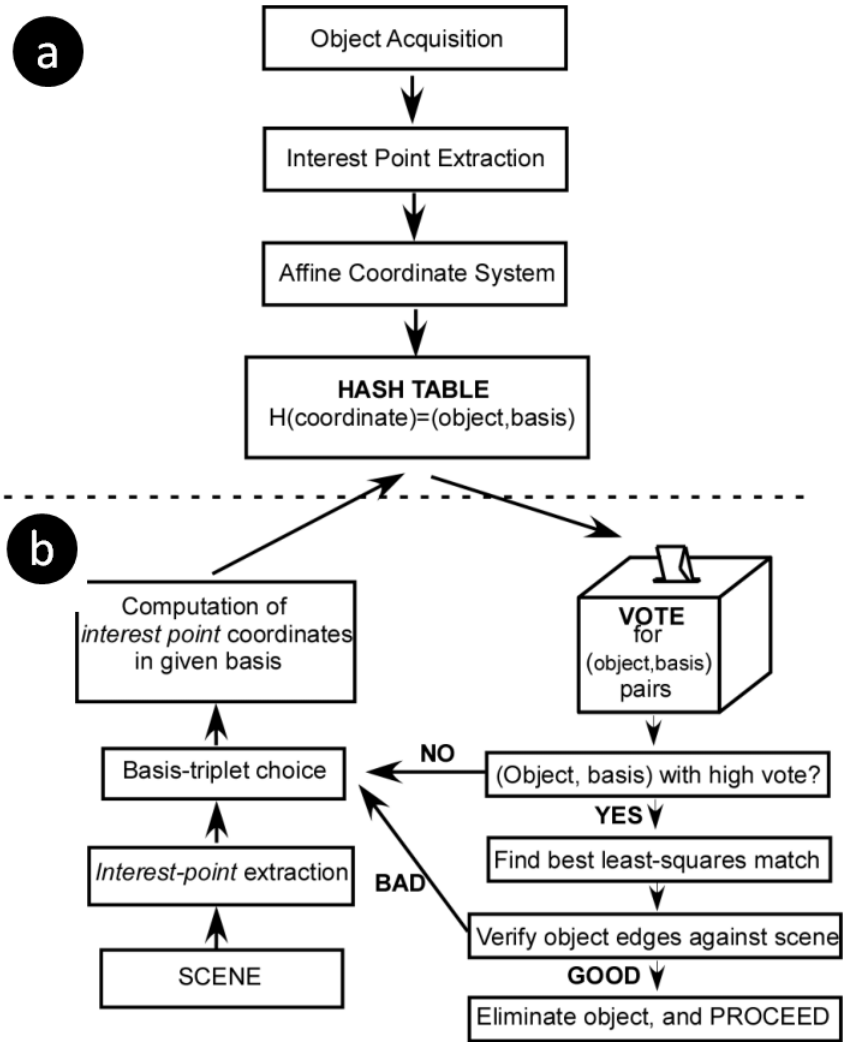


FIGURE 3.2: Geometric Hashing based object recognition algorithm. (a) offline or modeling stage. (b) online or recognition stage. image original link.

Bibliography

Abadi, Martín and David G Andersen (2016). "Learning to protect communications with adversarial neural cryptography". In: *arXiv preprint arXiv:1610.06918*.

Balasubramanian, Mukund and Eric L Schwartz (2002). "The isomap algorithm and topological stability". In: *Science* 295.5552, pp. 7–7.

Bay, Herbert et al. (2008). "Speeded-up robust features (SURF)". In: *Computer vision and image understanding* 110.3, pp. 346–359.

Belkin, Mikhail and Partha Niyogi (2003). "Laplacian eigenmaps for dimensionality reduction and data representation". In: *Neural computation* 15.6, pp. 1373–1396.

Bengio, Yoshua et al. (2009). "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1, pp. 1–127.

Calonder, Michael et al. (2010). "Brief: Binary robust independent elementary features". In: *European conference on computer vision*. Springer, pp. 778–792.

Chen, Mingxia et al. (2018). "Robust Semi-Supervised Manifold Learning Algorithm for Classification". In: *Mathematical Problems in Engineering* 2018.

Chopra, Sumit, Raia Hadsell, and Yann LeCun (2005). "Learning a similarity metric discriminatively, with application to face verification". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE, pp. 539–546.

Dai, Jifeng et al. (2016). "R-fcn: Object detection via region-based fully convolutional networks". In: *Advances in neural information processing systems*, pp. 379–387.

Darom, Tal and Yosi Keller (2012). "Scale-invariant features for 3-D mesh models". In: *IEEE Transactions on Image Processing* 21.5, pp. 2758–2769.

Ghodsi, Ali (2006). "Dimensionality reduction a short tutorial". In: *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* 37, p. 38.

Ghosh, Arnab et al. (2018). "Multi-Agent Diverse Generative Adversarial Networks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Girshick, Ross (2015). "Fast R-CNN Object detection with Caffe". In: *Microsoft Research*.

Goodfellow, Ian et al. (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*, pp. 2672–2680.

Goodfellow, Ian J et al. (2013). "An empirical investigation of catastrophic forgetting in gradient-based neural networks". In: *arXiv preprint arXiv:1312.6211*.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hinton, Geoffrey E and Sam T Roweis (2003). "Stochastic neighbor embedding". In: *Advances in neural information processing systems*, pp. 857–864.

Hotelling, Harold (1933). "Analysis of a complex of statistical variables into principal components." In: *Journal of educational psychology* 24.6, p. 417.

Huang, Jonathan et al. (2017). "Speed/accuracy trade-offs for modern convolutional object detectors". In: *IEEE CVPR*. Vol. 4.

Johnson, Andrew E (1997). "Spin-images: a representation for 3-D surface matching". In:

Karami, Ebrahim, Siva Prasad, and Mohamed Shehata (2017). "Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images". In: *arXiv preprint arXiv:1710.02726*.

Krig, Scott (2016). *Computer vision metrics*. Springer.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Kruskal, Joseph B (1964). "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis". In: *Psychometrika* 29.1, pp. 1–27.

Kulis, Brian et al. (2013). "Metric learning: A survey". In: *Foundations and Trends® in Machine Learning* 5.4, pp. 287–364.

Lamdan, Yehezkel and Haim J Wolfson (1988). "Geometric hashing: A general and efficient model-based recognition scheme". In:

Liu, Ming-Yu and Oncel Tuzel (2016). "Coupled generative adversarial networks". In: *Advances in neural information processing systems*, pp. 469–477.

Liu, Wei et al. (2016). "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer, pp. 21–37.

Lowe, David G (2004). "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2, pp. 91–110.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

Metz, Luke et al. (2017). "Unrolled generative adversarial networks". In: *ICLR 2017; arXiv preprint arXiv:1611.02163*.

Mika, Sebastian et al. (1999). "Kernel PCA and de-noising in feature spaces". In: *Advances in neural information processing systems*, pp. 536–542.

Murase, Hiroshi and Shree K Nayar (1995). "Visual learning and recognition of 3-D objects from appearance". In: *International journal of computer vision* 14.1, pp. 5–24.

Pearlmutter, Barak A and Lucas C Parra (1996). "A context-sensitive generalization of ICA". In: *Advances in neural information processing systems* 151.

Raducanu, Bogdan and Fadi Dornaika (2012). "A supervised non-linear dimensionality reduction approach for manifold learning". In: *Pattern Recognition* 45.6, pp. 2432–2444.

Rosten, Edward and Tom Drummond (2006). "Machine learning for high-speed corner detection". In: *European conference on computer vision*. Springer, pp. 430–443.

Roweis, Sam T and Lawrence K Saul (2000). "Nonlinear dimensionality reduction by locally linear embedding". In: *science* 290.5500, pp. 2323–2326.

Rublee, Ethan et al. (2011). "ORB: An efficient alternative to SIFT or SURF". In: *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, pp. 2564–2571.

Salakhutdinov, Ruslan and Geoff Hinton (2007). "Learning a nonlinear embedding by preserving class neighbourhood structure". In: *Artificial Intelligence and Statistics*, pp. 412–419.

Salimans, Tim et al. (2016). "Improved techniques for training gans". In: *Advances in Neural Information Processing Systems*, pp. 2234–2242.

Sammon, John W (1969). "A nonlinear mapping for data structure analysis". In: *IEEE Transactions on computers* 100.5, pp. 401–409.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Snell, Jake, Kevin Swersky, and Richard Zemel (2017). "Prototypical networks for few-shot learning". In: *Advances in Neural Information Processing Systems*, pp. 4077–4087.

Tran, Ngoc-Trung, Tuan-Anh Bui, and Ngai-Man Cheung (2018). "Dist-gan: An improved gan using distance constraints". In: *Computer Vision–ECCV 2018*. Springer, pp. 387–401.

Vincent, Pascal et al. (2008). "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 1096–1103.

Vinyals, Oriol et al. (2016). "Matching networks for one shot learning". In: *Advances in Neural Information Processing Systems*, pp. 3630–3638.

Wang, Chanpaul Jin, Hua Fang, and Honggang Wang (2016). "ESammon: A computationaly enhanced Sammon mapping based on data density". In: *Computing, Networking and Communications (ICNC), 2016 International Conference on*. IEEE, pp. 1–5.

Wolfson, Haim J and Isidore Rigoutsos (1997). "Geometric hashing: An overview". In: *IEEE computational science and engineering* 4.4, pp. 10–21.

Yang, Xin et al. (2006). "Semi-supervised nonlinear dimensionality reduction". In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 1065–1072.