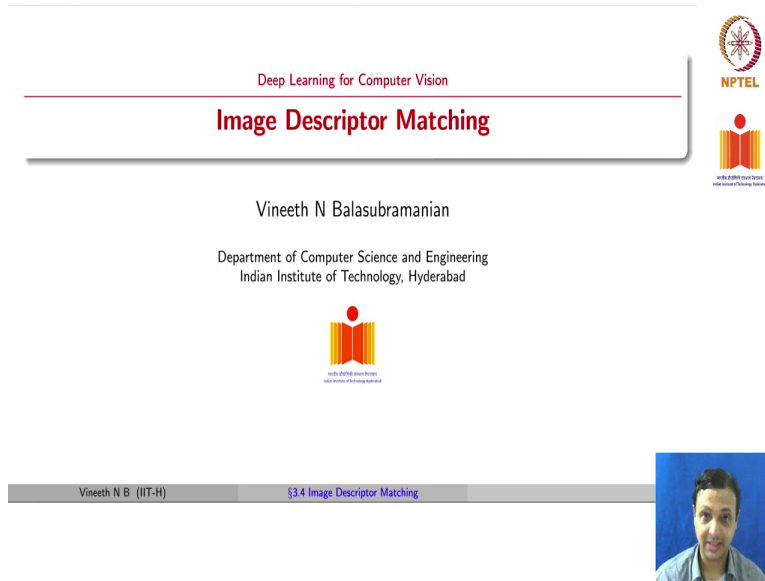


**Deep Learning for Computer Vision**  
**Prof. Vineeth N Balasubramanian**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Lecture 18**  
**Image Descriptor Matching**

(Refer Slide Time: 0:14)





The slide content includes the following elements:

- Top Header:** "Deep Learning for Computer Vision" in red text, followed by "Image Descriptor Matching" in a larger red font.
- Speaker Information:** "Vineeth N Balasubramanian" and "Department of Computer Science and Engineering, Indian Institute of Technology, Hyderabad" in black text.
- Logos:** NPTEL logo (top right) and IIT Hyderabad logo (center).
- Footer:** A grey bar at the bottom with "Vineeth N B. (IIT-H)" on the left and "§3.4 Image Descriptor Matching" in blue text on the right.
- Video Feed:** A small video window on the right side of the slide showing the speaker.

Last lecture we spoke about describing images using the bag-of-words approach or the VLAD approach. We will now take this forward and show how these descriptors can be used for matching between images.

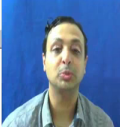
(Refer Slide Time: 00:39)



Acknowledgements

- Most of this lecture's slides are based on lectures of **Deep Learning for Vision** course taught by Prof Yannis Avrithis at Inria Rennes-Bretagne Atlantique

Vineeth N B (IIT-H) §3.4 Image Descriptor Matching

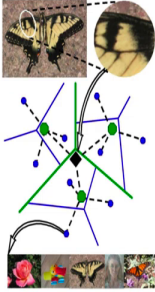


Before we go there, once again, an acknowledgement that these slides are taken from the excellent lectures of Professor Avrithis at Inria Rennes.

(Refer Slide Time: 00:47)

Review<sup>1</sup>


Hierarchical *k*-means and BoW:



- Apply hierarchical *k*-means and build a fine partition tree
- Descriptors descend from root to leaves by finding nearest node at each level
- Image represented by  $x_i = \sum w_j$  as in BoW
- Dataset searched by inverted files at leaves
- No principled way of defining  $w_j$  across levels
- Distortion minimized only locally; points can get assigned to leaves that are not globally nearest

<sup>1</sup>Nister and Stewenius, Scalable Recognition With a Vocabulary Tree, CVPR 2006

Vineeth N B (IIT-H) §3.4 Image Descriptor Matching



We also left behind one question from last time: since bag-of-words is inherently dependent on *k*-means to define its cluster centers, can we consider extensions of *k*-means to improve how bag-of-words performs. So one specific example could be hierarchical *k*-means which is an extension of *k*-means clustering algorithm, where the

cluster centers are organized in a hierarchical manner, starting from a root node, all the way to a set of leaf nodes. So it happens that this has been explored for bag-of-words two using a method known as vocabulary tree way back in 2006.

And what this method does is to take hierarchical k-means and build a final partition tree and now your image descriptors descend from each, from the root to one of the leaves in each level of the tree. So you have a bunch of cluster centers that you put together from all of the visual words that you have from different images. Recall that in bag-of-words, when you build your cluster centers, you pool all the images in your data together, you take all the features, the descriptors around the features, cluster them using a method, and then you take those clusters as your, what are known as codebook centers to which each key point is assigned.

Your image is finally represented as  $x_i$ , which is one of the elements in your image, in the descriptor of your image is given by  $w_i$  into  $n_i$ ,  $w_i$  is the weight of that particular node in the tree and  $n_i$  is the number of key points assigned to that particular cluster center or cluster centroid in the tree. So one evident thing here is it is difficult to know how  $w_i$  is given a value. One could argue that perhaps for levels down the tree, you must have a higher weight because they are a better match. One could also argue otherwise, depending on a particular setting there a high level match is perhaps more important than a low level match. It depends on the application, depends on what is important in a particular context.

So, which is a constraint of this method that there is no principle way of defining  $w_i$ , although you could come up with some heuristic on top of the method. The dataset is again searched using inverted file indices just the maybe we saw it in bag-of-words. And fundamentally there is an issue here which is that distortion is often minimized only locally, for example, around a particular cluster centroid. So any error that you make or any differences between images are only local with respect to that particular cluster centroid, distortion is not measured on a global sense. It is all with respect to each cluster centroid, across the occurrence of each cluster centroid in each image across two images that you are matching. That is how hierarchical k-means can be extended for

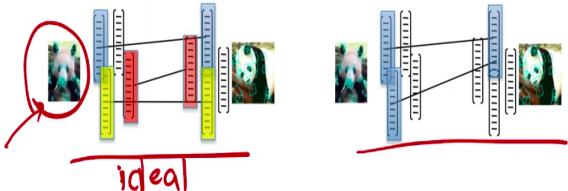
bag-of-words. So one could also consider other extensions of k-means and be able to use them in such methods to be able to describe images.

(Refer Slide Time: 04:28)

**Image Descriptor Matching: Options So Far**

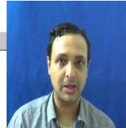
**Nearest Neighbor Matching:**

- Use each feature in a set to independently index into second set. Any problems you see?
- Ignores possibly useful information of co-occurrence  $\Rightarrow$  fails to distinguish between instances where an object has varying numbers of similar features since multiple features may be matched to a single feature in the other set



Source: Alberto Del Bimbo, UNIFI, Italy

Vineeth N B (IIT-H) §3.4 Image Descriptor Matching



Let us now talk about how you match descriptors from two different images in a more principled manner. Before we go there, let us try to assess what we know so far? One of the simplest methods that we have explored so far is nearest neighbor matching, where you have one image, which is a set of feature points, another image, which is a set of another feature points. At this time, we are not talking about any bag-of-words aggregation. It is simply a set of features in one image and a set of features in the other image.

We use each feature in a set and its corresponding descriptor to independently index into a feature from the second set. We simply do a nearest neighbor matching based on the descriptor of the feature in each of these images. Can you think of what could be a limitation of such an approach?

An inherent limitation of this approach is that you could be ignoring useful information of co-occurrence. So it is possible that there could be one image where there could be multiple instances of the same feature. Think of, say, spots in a leopard or stripes in a

zebra or so on and so forth. And you may be mapping a single feature in one image to multiple different features in the other image because they all look similar.

For example, a spot on a leopard in one image could get mapped to multiple spots on a leopard in a second image. You ideally would not want to have, want this to happen. You would want each spot on a leopard to get mapped to be one spot on the leopard in the other image and another spot to get mapped to another spot and so on and so forth.

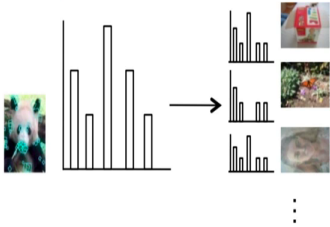
Or the example that you see on the slide here, which is a giant panda, which again has some features which repeat in its structure, which could get mapped to the same instance in the second image, which is what you see illustratively here. This will be the ideal setting where each feature gets mapped to the same, to an independent feature in the second image and this could be another scenario where two different instances of the same feature get mapped to the same feature on the second image which is not desirable. This could lead to some problems when you use a nearest neighbor matching approach.

(Refer Slide Time: 07:01)

**Image Descriptor Matching: Options So Far**


**Bag-of-Words Matching:** Any glaring limitation?

- Can only compare entire images to one another and does not allow partial matchings
- This implies an **all-all** matching; it is often preferable to have a **one-one** matching instead



Source: Alberto Del Bimbo, UNIFI, Italy

Vineeth N B (IIT-H) §3.4 Image Descriptor Matching



We have also seen the bag-of-words matching approach so far. Any glaring limitation that you see of this approach, an evident limitation is that the bag-of-words approach is limited to a full image matching, not a partial matching, because you are going to be looking at the histogram of the occurrences of a cluster centroid in image one versus such

a histogram for image two, you would say two images match only when the complete histograms are fairly close to each other. So if only a part of the second image matches the first dimension, these histograms will not match and you will not get a good match in this kind of setting. So in other words, you could say the bag-of-words is an all-all matching, but we ideally want some sense of a one-to-one matching of one part of the image matching with another part, so that we can still match some images which could be partially close to one another. Ideally speaking, the nearest neighbor matching was a one-to-one matching approach, but it has its own limitations.

(Refer Slide Time: 08:16)

### Generalizing Descriptor Matching using Kernels<sup>2</sup>

- Consider an image described by a set of  $n$  descriptors (features)  $X = \{x_1, x_2, \dots, x_n\}$ , each of  $d$  dimensions
  - Descriptors typically quantized using  $k$ -means clustering
  - Quantizer  $q: R^d \rightarrow C \subset R^d$  maps each descriptor to a representative descriptor, a.k.a **visual word**
  - $C = \{c_1, c_2, \dots, c_k\}$  is a codebook consisting of  $k$  visual words.
- To compare two image representations  $X$  and  $Y$ , let us define a general family of matching kernels:

$$K(X, Y) = \gamma(X) \gamma(Y) \sum_{c \in C} M(X_c, Y_c) \quad (1)$$

where  $X_c = \{x \in X : q(x) = c\}$  is the set of descriptors assigned to the same visual word,  $M$  is a within-cell matching function, and  $\gamma$  is a normalization function

<sup>2</sup>Tolias et al, To Aggregate or Not to aggregate: Selective Match Kernels for Image Search, CVPR 2013



Now let us try to generalize how you can do this descriptor matching using a method that you would have seen in machine learning which is kernels. You may have heard about kernels in support vector machines. So we are going to use a similar idea here to be able to generalize matching between descriptors. Recall that even in support vector machines or for that matter any other machine learning algorithm works where kernels can be used. Kernels are a sense of similarity between two data points. It is the same principle that is being used here also.

To be able to do that let us define the setting so far. So you have an image described by end descriptors let us say. So  $X$  is an image given by  $x_1, x_2, \dots, x_n$  where each of these

is a  $d$  dimensional vector. Remember this could just be the cluster centroids or could be individual features.

And looking at the bag-of-words example, these descriptors are typically quantized using  $k$ -means clustering or for that matter any other clustering method. They are quantized, which means you simply do not take all the features in an image. You try to see which cluster center they belong to and only have a count in that particular cluster center as a representation of that particular feature or that particular cluster center. This quantization function is given by  $q$ , which takes us from  $R^d$  to a subset of  $C$  of  $R^d$ , where  $C$  is called a codebook, which is given by  $c_1$  to  $c_k$ .

So there are  $k$  possible cluster centroids which, as I just mentioned, you get by doing a  $k$ -means clustering on the features from all images. And in a single image, you try to see, you use a quantizer function, which takes you from every feature to one of these cluster centroids which is nearest to it. This is the setting.

Now let us try to define the kernel. So the kernel now is given by given two images,  $X$  and  $Y$ , the matching kernel  $K(X, Y) = \gamma(X)\gamma(Y)$  which as we will see soon are normalization functions, we will see why we need this in a moment, into summation over all the cluster centers that you have  $\sum M(X_c, Y_c)$ , where  $M$  is the within-cell matching function.

So it is the,  $M$  is the matching kernel function that you are talking about and that happens for each, the occurrences of each cluster centroid in one image and the occurrences of that cluster centroid in the second image. So we still have to define what  $M$  is. We will see a few examples of that as we go forward.

So  $\gamma(X)\gamma(Y)$  is required here because you do not want to be biased by the presence of number of features in a given image. For example, it is possible that you may detect say a 1,000 features in one image and just 200 in the other. If you would simply do a summation you would always be biased by an image that has a lot of features because the count would go up and which may not really be a perfect match.

So the  $\gamma(X)\gamma(Y)$  are normalization functions, where you can divide by the total number of features in the image so that the matching is not biased by the, simply the number of features in an image.



(Refer Slide Time: 12:01)

### Bag of Words Matching<sup>3</sup>

Recall: BoW model characterizes an image solely by visual words

o Cosine similarity in BoW model can be defined by defining  $M$  as:

$$M(X_c, Y_c) = \sum_{x \in X_c} \sum_{y \in Y_c} 1$$

*Image Credit: Fei-Fei, Fergus and Torralba, Recognizing and Learning Object Categories, CVPR 2007 Tutorial*  
<sup>3</sup>Jegou et al, Aggregating local descriptors into a compact image representation, CVPR 2010  
 Vineth N B (IIT-H) §3.4 Image Descriptor Matching

Now let us try to see a few examples of how  $M$  would look in examples that we have seen so far. So if you turn to the bag-of-words matching, bag-of-words matching is in a way a cosine similarity between the cluster centers in the two images. You could define that whatever we have seen in bag-of-words similarity so far can be defined by a matching kernel which is given by, you take  $X_c$  which is one of your codebook elements or your cluster centroids and you count how many occurrences of that are there in image  $X$ , how many occurrences are there in image  $Y$  and you simply add them all.

Rather for one particular codebook entry  $c_3$ , if you had 10 such features in image  $X$  that correspond to  $c_3$  and three such features in image  $Y$  that correspond to  $c_3$ , the corresponding matching kernel is simply  $10 \times 3$  here, which becomes 30. You simply count that over a double summation. So that is simply a bag-of-words model. Remember that the normalization function, as I said, would take care of normalizing by the total number of features in the image itself. But this is what  $M$  is defined as.



(Refer Slide Time: 13:24)

### Hamming Embedding for Matching<sup>4</sup>

- In addition to being quantized, each descriptor  $x$  is binarized as  $b_x$ .
- Score is computed between all pairs of descriptors assigned to the same visual word as:

$$M(X_c, Y_c) = \sum_{x \in X_c} \sum_{y \in Y_c} \mathbb{1}[h(b_x, b_y) \leq \tau]$$

where  $h(\cdot, \cdot)$  is the Hamming distance between two binary vectors, and  $\tau$  is a threshold to count matched pairs

<sup>4</sup>Jegou et al, Aggregating local descriptors into a compact image representation, CVPR 2010  
Vineeth N B (IIT-H) §3.4 Image Descriptor Matching



You could extend this to another approach known as hamming embedding for matching, where if you assume that each descriptor can be binarized in some way, for example, you could take a descriptor and simply say that anything greater than a threshold is one and anything less than a threshold is zero, you could binarize any descriptor for that matter.

Then you compute your matching kernel as it is, again, similar to your bag-of-words kernel. The only difference now is you are only going to count the number of instances where the hamming distance between  $b_x$  and  $b_y$  is going to be less than a threshold. This is simply your hamming embedding. So the hamming,  $h$ , is the hamming distance between the two binary vectors and  $\tau$  is a threshold that you have to specify to be able to get a matching kernel in this particular setting.

(Refer Slide Time: 14:25)

### VLAD Matching<sup>5</sup>

- **Recall:** For each visual word, VLAD performs *pooling* by constructing a vector representing the sum of residuals as:

$$V(X_c) = \sum_{x \in X_c} r(x) \quad \text{where} \quad r(x) = x - q(x)$$



<sup>5</sup>Jegou et al, Aggregating local descriptors into a compact image representation, CVPR 2010  
Vineeth N B (IIT-H) §3.4 Image Descriptor Matching



You can also define VLAD matching in the same framework. Remember, again, that VLAD is similar to bag-of-words, the only difference is you do not count how many features belong to a codebook element or a cluster center. You rather obtain all the features that belong to a codebook element or a cluster center and rather count the residual vector.


Recall again, in VLAD you have a cluster center. For example, you could have two cluster centers. You have a set of features that are closest to this particular cluster center. You have another set of features which are closest to this particular cluster center. So you take the difference between them which is going to be a residual vector and you add up all the residual vectors that belong to one particular cluster center and that becomes the representation for this cluster center. And similarly, you would do this for other cluster centers.

(Refer Slide Time: 15:30)

**VLAD Matching<sup>5</sup>**

- Recall: For each visual word, VLAD performs *pooling* by constructing a vector representing the sum of residuals as:
 
$$V(X_c) = \sum_{x \in X_c} r(x) \quad \text{where} \quad r(x) = x - q(x)$$
- A  $d \times k$  vector is constructed for an image  $X$  as follows:
 
$$V(X) = (V(X_{c_1}), V(X_{c_2}), V(X_{c_3}), \dots, V(X_{c_k}))$$
- Matching kernel now defined as:
 
$$M(X_c, Y_c) = V(X_c)^T V(Y_c) = \sum_{x \in X_c} \sum_{y \in Y_c} r(x)^T r(y)$$

<sup>5</sup>Jegou et al, Aggregating local descriptors into a compact image representation, CVPR 2010  
Vineeth N B (IIT-H)      §3.4 Image Descriptor Matching



Now, if this was the representation, how do you do the matching kernel? So the matching kernel is given by the entire representation of image  $X$  is going to be a set or an entire vector represented by  $(V(X_{c_1}), V(X_{c_2}), V(X_{c_3}), \dots, V(X_{c_k}))$  That is going to be the representations corresponding to each of your codebook elements. And your matching kernel is now given by  $V(X_{c_1})^T V(Y_c)$ , which is an inner product between the representation for the  $c^{th}$  codebook entry in  $X$  and the  $c^{th}$  codebook entry in  $Y$ . But this is simple to expand, because  $V(X_c)$  is a summation over all the  $X$ s that belong to that particular code entry and their corresponding residuals.

So you expand  $V(X_c)$  using a summation. Similarly,  $V(Y_c)$  using its summation and you now have the new matching kernel as a summation over all the elements or the, all the features that belong to that codebook entry in image  $X$  and a summation over  $Y$  for the same codebook entry and inside you are going to have  $r(x)^T r(y)$ , so the residuals are aligned with each other.

Rather, if you try to understand the intuition here, you are trying to say that if you have a cluster center, say  $c_3$  in image  $X$ , the same cluster center  $c_3$  in image  $Y$ . Let us say you

have three features in image X belonging to this cluster center. Similarly say three features belonging to this cluster center in image Y. You are going to take one of these residuals and see how the other residual matches with this residual. If the two residuals match, you are going to get a high matching score. Rather, if even how the features were configured around the cluster center match between the two images, it is a better match.

(Refer Slide Time: 17:41)



### Aggregated Selective Match Kernel (ASMK)<sup>6</sup>

- ASMK is a combination of two borrowed ideas:
  - Non-linear selective function (from Hamming Embedding)
  - Pooling Residuals (from VLAD)
- Matching kernel  $M$  is expressed as:
 
$$M(X_c, Y_c) = \sigma_\alpha(\hat{V}(X_c)^T \hat{V}(Y_c))$$


where  $\sigma_\alpha$  is a non-linear function given by:

$$\sigma_\alpha(u) = \begin{cases} \text{sign}(u)|u|^\alpha & \text{if } |u| > \tau \\ 0 & \text{otherwise} \end{cases}$$

and  $\hat{V}(X_c) = V(X_c) / \|V(X_c)\|$  and  $V(X_c)$  is VLAD representation discussed earlier

<sup>6</sup>Tolias et al, To Aggregate or Not to aggregate: Selective Match Kernels for Image Search, CVPR 2013  
 Vineth N B (IIT-H) §3.4 Image Descriptor Matching



A more general way of combining all of these ideas was known as the aggregated selective match kernel or ASMK which combines a few ideas that we have seen so far. It combines the non-linear selective function that we saw with hamming embedding. We will see that in a moment and also it combines ideas from VLAD.

So the way this method works is you take VLAD, which is what you see as the argument inside and you normalize the VLAD vectors, which is what you define as  $\hat{V}$ . So  $\hat{V}$ , what you see at the bottom of the slide is given by  $V(X_c) / \|V(X_c)\|$ , effectively you are trying to make the VLAD vector which is the sum of all residuals into a unit norm vector. And you now take an inner product between the VLAD vectors, very similar to what we saw in the previous slide. Because this is an inner product, the output of this is going to be a scalar.

Now you use a non-linear selective function  $\sigma_\alpha$  of this scalar to get your final matching function. What is this  $\sigma_\alpha$ ? The  $\sigma_\alpha$  is defined as for any input  $u$ ,  $\sigma_\alpha$  is defined as the  $sign(u) |u|^\alpha$ . If  $u$  is greater than a threshold  $\tau$ , remember the inner product is a measure of similarity. Higher the value, the better for  $u$ . So if  $u$  is greater than a threshold, you may want to weigh things a bit more which you can control using  $\alpha$ . And if  $u$  is less than a threshold, it is zero, very similar to the hamming distance. We say that if the hamming distance was less than a threshold, we will count it more. If not, we would not count it. This is a very similar idea, although we use VLAD vectors for achieving the same goal. So you can see here that if  $\alpha$  is 1, this is just  $u$  itself. If  $\alpha$  is 1,  $sign(u) |u|$  will be  $u$  itself.

(Refer Slide Time: 20:01)

**Aggregated Selective Match Kernel (ASMK)**

- ASMK matching with different values of distance threshold and selectivity parameter
- Yellow corresponds to 0 similarity and red to maximum similarity per image pair, as defined by selective function
- Larger selectivity drastically down-weights false correspondences
- This replaces hard thresholding in the Hamming Embedding method

Vineeth N B (IIT-H)      §3.4 Image Descriptor Matching

Let us see a few illustrations of this idea to make this clearer. So this is a few illustrations of different choices of alpha and tau values. So on the top left here, you see  $\alpha$  is 1, which as, we just said, which is  $u$  itself. It is simply the dot product of the normalized VLAD vectors and  $\tau$  in this case is zero. So anything greater than zero, you are going to consider that as  $u$  itself to be the as the, the score to be  $u$  itself.

So you can see here, in this case, yellow corresponds to say zero similarity and red corresponds to maximum similarity per image pair. So there are few features that do not

match at all and there are a few features that match very well and all of them are shown in the top left image.

If you see the top right, you can see here, again  $\alpha$  is equal to 1, but  $\tau$  is equal to 0.25. You can see now that a lot of the yellows have come down. You are now saying that that score  $u$  must be at least 0.25 for us to consider that to be a match. And you see now that many of these false matches disappear because anything, any low score is now disregarded.

The bottom row shows an example where  $\alpha$  is equal to 3. Where you see again, when  $\alpha$  is equal to 3, remember that because you are normalizing your VLAD vectors,  $\alpha$  being high is going to reduce the value because  $u$  is going to be a value lying between 0 and 1, because you have normalized the vectors, it is likely to be a value between 0 and 1. So if  $\alpha$ , which is the exponent of  $u$  is high, the values are going to go even smaller and that is what you see here on the left, where do you see that some of these reds have disappeared, because they have gone closer to 0.

And on the right, you see a similar thing where when  $\alpha$  is equal to 3 and  $\tau$  is equal to 0.25, again you get a few more yellows, which could be smaller values at this point in time, because  $u$ , the exponent  $u$ , exponent  $\alpha$  which is 3 here may have reduced the values because  $u$  lies between 0 and 1.

So you can see here that larger selectivity down-weights false correspondences, which is what we see here with tau is equal to 0.25 on both the images on the right. And this entire approach replaces the hard thresholding that we have in hamming embedding and gives a different way of going about a similar approach.

(Refer Slide Time: 22:41)

### Aggregated Selective Match Kernel (ASMK)



ASMK Example: Each visual word is drawn with a different color



Here is another illustration of results after applying the ASMK method. Where you see here that each of these colors in these different images correspond to the same visual word. It is, so the green or the yellow or the blue is the same visual word occurring in different images. So you can see here that if you take any particular example, for example, if you take, say the pink or the red, you would see that the pink or the red visual word corresponds to some corner or some pointed corner in each of these images.

(Refer Slide Time: 23:18)

### Efficient Match Kernels<sup>7</sup>



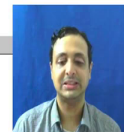
- Instead of threshold-based matching functions (as used in HE), we can use a continuous function  $\kappa(x, y)$  and avoid using computationally intensive codebooks:

$$K(X, Y) = \gamma(X)\gamma(Y) \sum_{x \in X} \sum_{y \in Y} \kappa(x, y)$$

- Such a function  $K(X, Y)$  can be decomposed into an inner product of  $\Phi(X)$  and  $\Phi(Y)$
- To do that, we learn a low-dimensional feature map  $\phi$  such that  $\kappa(x, y) = \phi(x)^T \phi(y)$  and:

$$K(X, Y) = \left( \gamma(X) \sum_{x \in X} \phi(x) \right)^T \left( \gamma(Y) \sum_{y \in Y} \phi(y) \right) = \Phi(X)^T \Phi(Y)$$

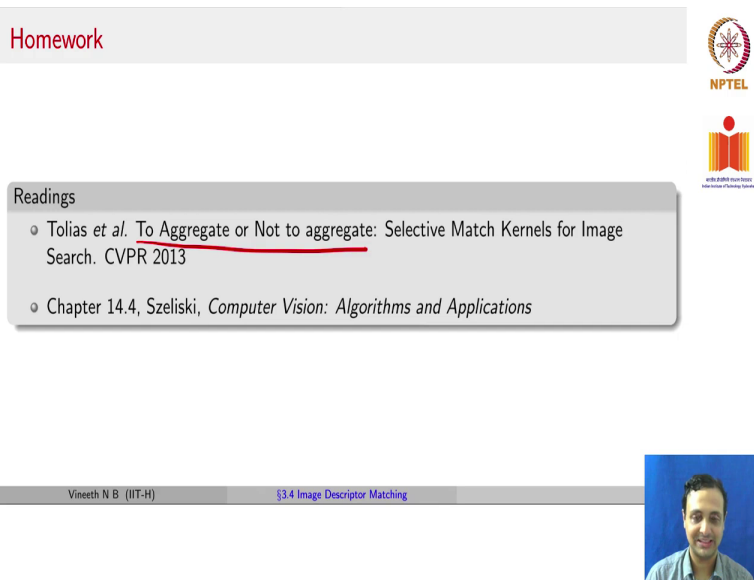
<sup>7</sup>Bo and Sminchisescu, Efficient Match Kernels between Sets of Features for Visual Recognition, NeurIPS 2009



All of these kernels can be generalized into efficient match kernels, where you could define this kernel as some continuous function  $K(X, Y)$  and avoid using any codebooks for that matter at all. Because codebooks can be computationally intensive to compute and then compute the residuals and so on and so forth, you could directly impose a kernel function between the individual features in one image and the individual features in the other image.

Once again, very similar to how kernel functions were used in support vector machines or other machine learning algorithms which had a kernel competent to it. Ideally you would want this  $K(X, Y)$  can be decomposed into  $\phi(x)\phi(y)$ , where  $\phi$  is the representation of each feature in a different space. Then you would have your final kappa to be some normalization of  $X$  into the summation of all of your representations of  $x$  for each of these, for all your features transpose a similar aggregation for  $Y$ . This is a generalization of these methods that we have seen so far.

(Refer Slide Time: 24:42)



Homework

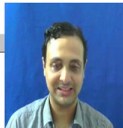
Readings

- o Tolias *et al.* To Aggregate or Not to aggregate: Selective Match Kernels for Image Search. CVPR 2013
- o Chapter 14.4, Szeliski, *Computer Vision: Algorithms and Applications*

NPTEL

© IIT Bombay

Vineeth N B (IIT-H) §3.4 Image Descriptor Matching



That concludes this lecture. So for more readings, this particular paper to aggregate or not to aggregate summarizes these kernels very well. Please do read through it when you get a moment and chapter 14.4 from Szeliski's book would be your other reference for this lecture.