



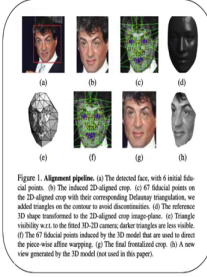
**Deep Learning for Computer Vision**  
**Professor. Vineeth N Balasubramanian**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Lecture No. 49**  
**CNNs for Human Understanding: Faces – Part 02**

(Refer Slide Time: 00:10)

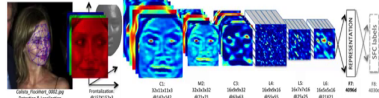



**DeepFace: Identification<sup>2</sup>**

- o **Step 1:** Face localization, fiducial point detection and alignment  $\Rightarrow$  frontal crop of face
- o **Step 2:** Frontal crop passed for identification to deep CNN model with K-way softmax (multi-class classification)



**Figure 1. Alignment pipeline.** (a) The detected face, with 4 initial fiducial points. (b) The detected 2D aligned crop. (c) 67 fiducial points on the 2D aligned crop with their corresponding Delaunay triangulation, we added triangles on the corners to avoid distortions. (d) The reference 3D shape transformed to the 2D aligned crop image-plane. (e) Triangled visibility  $w \times 1$  to the fitted 3D camera, darker triangles are less visible. (f) The 67 fiducial points induced by the 3D model that are used to direct the piece-wise affine warping. (g) The final frontalized crop. (h) A new view generated by the 3D model (not used in this paper).



**Figure 2. Outline of the DeepFace architecture.** A first set of a single convolution pooling convolution filtering on the rectified input, followed by three fully-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.



<sup>2</sup>Taigman et al, DeepFace: Closing the Gap to Human-Level Performance in Face Verification, CVPR 2014  
 Vineeth N. B. (IIT-H) | 37.4 CNNs for Human Understanding: Faces | 14 / 37



Now, we will move on to some of the recent efforts that have used CNNs to perform face identification and verification. One of the first efforts in recent years since the success of AlexNet was DeepFace, which was published in CVPR of 2014, which first performed a pre-processing step, which included Face Localization, Fiducial Point Detection and alignment to get a frontal crop of the face. This image on the left shows the steps that were involved to obtain the frontal crop.

Given the input image, the first step is to detect the face and crop out the face from the image. Then, about 67 fiducial points are detected on the 2D crop and they are triangulated. These 67 fiducial points are different markers on the face, such as tip of the nose, corners of the eyes, corners of the lips, so on and so forth. There are existing methods that can do that. Once this is done, these are then cast on to a 3D model, onto which this particular face image is projected. And that 3D model is then normalized into a 2D face image with its corresponding fiducial points, which is converted to a frontal crop of the image.

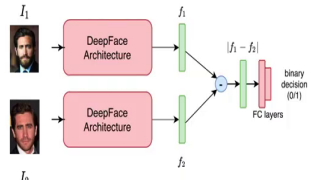
And finally, since we now have the 3D model of the some, same person's face, you could also generate other poses of the same person. So, the frontal crop that we talk about here in Step 1, is this image g, which you can see is fairly well normalized when you compare to the image

b. This frontal crop is passed on to the deep CNN model, deep CNN architecture with a K-way softmax for differentiating between K different classes using the standard soft, trained using the standard softmax and cross-entropy loss.

(Refer Slide Time: 02:32)





### DeepFace: Verification (Siamese Networks)



- Representation learned from identification used for verification  $\implies$  freeze classification parameters
- Given a face image  $I$ ,  $f = G(I) \implies$  penultimate layer representation for  $I$
- Network trained by taking absolute difference between features, followed by a fully connected head
- Distance induced:  

$$d(f_1, f_2) = \sum_i \alpha_i |f_1[i] - f_2[i]|$$
 where  $\alpha_i$  are trainable parameters
- **Output:** Binary decision (same/not same)



Vineeth N B. (IIT-H)
§7.4 CNNs for Human Understanding: Faces
15 / 37

For the verification part, DeepFace used a Siamese Network kind of an approach, where the representation learned from the identification task is frozen for verification. So, these architecture parameters are frozen. Given a face image  $I$  and its corresponding image in the record, which the person claims to be, you get 2 representations  $f_1$  and  $f_2$ , which is  $G(I_1)$  and  $G(I_2)$ . The network is then trained, the rest of the network is then trained by taking an absolute difference between  $f_1$  and  $f_2$  followed by a fully connected layer and then your final binary decision as Sigmoid Activation Function.

So, the distance induced in this particular scenario is  $|f_1[i] - f_2[i]| \alpha_i$ , where alpha is a weight learned in this particular layer. And the final output is then a match or not match or a same or a not same. This approach that was proposed in DeepFace, which was taken from the Siamese Network idea of the 90s has since been expanded in several ways.

(Refer Slide Time: 03:52)



## Contrastive Loss



- Based on metric learning paradigm
- Used to learn distinctive discriminative feature representations for various downstream computer vision tasks
- **Goal:** Map the input to embedding space where the distance between points corresponds to "semantic similarity" between input points
- Various formulations: Pairwise Contrastive Loss, Ranking Loss, Triplet Loss

We will look at some of these to build verification systems



Vineeth N B. (IIT-H)



§7.4 CNNs for Human Understanding: Faces

16 / 37

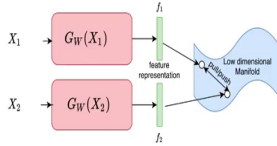
And one of the primary ideas, which has been used is the notion of what is known as Contrastive Loss. Contrastive loss is a loss that is used for training the neural network based on the paradigm of metric learning in machine learning. Metric learning methods in machine learning attempt to learn a distance metric, rather than use a Euclidean distance metric, or a cosine distance metric. So, the same idea is now used to learn distinctive discriminative feature representations for the task that you want to use these representations for.

So, in the face verification scenario, our objective is to map the input into an embedding space where the distance between points (are) correspond to semantic similarity. Euclidean distance could do it, but if we could learn a distance that does this more effectively, that would be more useful, and that is what most of these methods try to do. There have been various formulations, such as Pairwise Contrastive Loss, Ranking Loss, Triplet Loss, so on and so forth. We will see some of these over the rest of this lecture.


(Refer Slide Time: 05:15)

### Pairwise Contrastive Loss



- Hadsell<sup>3</sup> et al introduced an approach to learn a mapping
  - invariant to complex transforms
  - “similar” points in input space are mapped to nearby points on a low-dimensional manifold.
  - capable of consistently mapping new points (unseen during training)
- **Objective:** Learn  $W$  such that  $D_W(X_1, X_2) = \|G_W(X_1) - G_W(X_2)\|_2$  approximates the “semantic similarity” of inputs

 <sup>3</sup>Hadsell et al, Dimensionality Reduction by Learning an Invariant Mapping, CVPR 2006  
Vineeth N B. (IIT-H) §7.4 CNNs for Human Understanding: Faces 17 / 37

Pairwise contrastive loss came from a work in 2006, introduced by Hadsell, where an approach to learn a mapping invariant to complex transforms (for int) was introduced. And as we just said, the idea was to make similar points close to each other on the embedding manifold. So, this representation that you get as output of your CNN is also called an embedding. And if we now assume that those low dimensional embeddings, generally these embeddings have a lower dimensionality than the image itself, the image could be a  $200 \times 200$  image, whereas the embedding could just be a 1000-dimensional vector or a 4000-dimensional vector.

We ideally want these embeddings that lie on, say, 1000-dimensional manifold or a 4000-dimensional manifold to be similar to each other for similar entities, and far away from each other for dissimilar entities. So, our goal here is to learn  $W$ , the weights of the neural network in a way in which the distance or  $\|G_w(X_1) - G_w(X_2)\|_2$ , given  $X_1$  and  $X_2$  are inputs to approximate the Semantic Similarity of the inputs. Let us see how this is done.

(Refer Slide Time: 06:42)



## Pairwise Contrastive Loss

### Algorithm

The algorithm first generates the training set, then trains the machine.

**Step 1:** For each input sample  $\vec{X}_i$ , do the following:

(a) Using prior knowledge find the set of samples  $S_{\vec{X}_i} = \{\vec{X}_j\}_{j=1}^p$ , such that  $\vec{X}_j$  is deemed similar to  $\vec{X}_i$ .

(b) Pair the sample  $\vec{X}_i$  with all the other training samples and label the pairs so that:  
 $Y_{ij} = 0$  if  $\vec{X}_j \in S_{\vec{X}_i}$ , and  $Y_{ij} = 1$  otherwise.

Combine all the pairs to form the labeled training set.

**Step 2:** Repeat until convergence:

(a) For each pair  $(\vec{X}_i, \vec{X}_j)$  in the training set, do

i. If  $Y_{ij} = 0$ , then update  $W$  to decrease

$$D_w = \|G_w(\vec{X}_i) - G_w(\vec{X}_j)\|_2$$

ii. If  $Y_{ij} = 1$ , then update  $W$  to increase

$$D_w = \|G_w(\vec{X}_i) - G_w(\vec{X}_j)\|_2$$

Let  $\mathbf{x}_1, \mathbf{x}_2 \Rightarrow$  be a pair of high-dimensional input vectors

$y$  a binary label assigned to this pair:

$$\begin{cases} y = 0 & \text{if } \mathbf{x}_1, \mathbf{x}_2 \text{ are similar} \\ y = 1; & \text{otherwise} \end{cases}$$

Given  $(y, \mathbf{x}_1, \mathbf{x}_2)$  are labeled pairs of data,

**Pairwise Contrastive Loss,**

$L_{contrastive}(W, y, \mathbf{x}_1, \mathbf{x}_2)$  given by:

$$\frac{1-y}{2} D_w^2 + \frac{y}{2} \max(0, m - D_w^2)$$

where  $m > 0$  is a margin which defines a radius around  $G_w(\mathbf{x})$



Vineeth N B. (IIT-H)

§7.4 CNNs for Human Understanding: Faces

18 / 37

So, the overall algorithm's ideology can be given by this algorithm environment here. Given a training data set, you first identify the set of samples that are similar to each other and put them in a set, say,  $S_{X_i}$ , given an input  $X_i$ , we identify the samples  $X_j$  that are similar to  $X_i$ . In the face verification context, it could be other pictures of the same person. We also say that  $Y_{ij}$ , which is given  $X_i$  and  $X_j$  would be 0 if these points are similar. Similarly, you identify dissimilar points, which could be images of 2 different people, where you give a label  $Y_{ij} = 1$ . Now, what do we want to do when we train?


For each pair  $(X_i, X_j)$  in the training set, if  $Y_{ij} = 0$ , that means they are similar images, we then update  $W$  to decrease the distance  $\|G_w(X_i) - G_w(X_j)\|_2$ . And if  $Y_{ij} = 1$ , that is dissimilar images, then we would like to increase this distance  $D_w$ . Let us see now how this is done using a single loss function. So, given  $X_1$  and  $X_2$  be a pair of high dimensional input vectors, let  $y$  be a binary label given to be 0 if  $X_1$  and  $X_2$  are similar, and 1 otherwise.

Then the pairwise contrastive loss,  $L_{contrastive}$  is given by  $\frac{1-y}{2} D_w^2 + \frac{y}{2} \max(0, m - D_w^2)$ . The first term, as you can see, gets activated only when  $y = 0$ . And the second term gets activated only when  $y = 1$ . When  $y = 0$ , the images are similar. So, you are minimizing  $D_w^2$

, which is what we would like to do. And when  $y = 1$ , you have dissimilar images as input, then you are minimizing the  $\max(0, m - D_w^2)$ .

Let us analyse this.  $m$  is a user defined margin, that you want the dissimilar images to at least be separated by a certain number, 10 units or 20 units in your manifold, so on and so forth. And this states that if  $D_w^2 > m$ , then the second quantity will become negative, the max would become 0, and you are saying that there is nothing to do.  $D_w^2$  or the distance between the dissimilar images is already greater than my user defined margin  $m$ . So, this term would disappear. However, if  $D_w^2 < m$ , in that scenario, you would be minimizing  $m - D_w^2$ , which is equivalent to maximizing  $D_w^2$  or the distance between these dissimilar images.

(Refer Slide Time: 10:14)



### DeepID2<sup>4</sup>

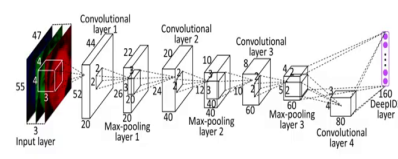
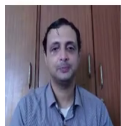


Figure 1: The ConvNet structure for DeepID2 extraction.

- Trains a deep CNN to jointly perform identification and verification
- **Identification task:** Increases inter-personal variations by pushing features from different identities apart
- **Verification task:** Reduces intra-personal variations by pulling features from same identity together



<sup>4</sup>Sun et al, Deep Learning Face Representation by Joint Identification-Verification, NIPS 2014

Vineeth N B (IIT-H) §7.4 CNNs for Human Understanding: Faces 19 / 37

This idea is used in the subsequent network DeepID2, which was proposed in NIPS of 2014. So this, similar to DeepFace, trains a deep CNN to jointly perform identification and verification, where it can be argued that the identification task increases inter-personal variations, you are trying to separate different identities apart, images of different identities or embeddings of different identities. And the verification task reduces intra-personal variations by bringing together features or embeddings of the same identity. How is this implemented?

(Refer Slide Time: 10:59)



## DeepID2



- **Cross-entropy loss** for training identification parameters  $\theta_{id}$
- **Pairwise contrastive loss** for learning verification parameters  $\theta_{ve}$

### Identification Loss

$$-\sum_{i=1}^n p_i \log \hat{p}_i = -\log \hat{p}_t$$
 $f$ : DeepID2 feature vector  
 $t$ : target class  
 $\theta_{id}$ : softmax layer parameters  
 $p_i$ : target probability distribution  
 $\hat{p}_i$ : is predicted probability distribution

### Verification Loss

$L_{contrastive}(\theta_{ve}, y_{ij}, f_i, f_j)$   
 $f_i, f_j$ : DeepID2 feature vectors for face images under comparison  
 $\theta_{ve}$ : verification loss parameters  
 $y_{ij} = 1 \implies f_i$  and  $f_j$  are from same identity  
 $y_{ij} = -1 \implies$  different identities



Vineeth N B. (IIT-H)

§7.4 CNNs for Human Understanding: Faces

20 / 37

In DeepID2, cross-entropy loss is used for the identification parameters and pairwise contrastive loss is used for learning the verification parameters. So, you have  $\theta_{id}$ , which correspond to the weights of the identification module and  $\theta_{ve}$ , which correspond to the verification module. The identification loss is given by cross-entropy as you can see here, where  $\theta_{id}$  or the Softmax layer parameters, the last layer parameters for identification, and similarly, the pairwise contrastive loss, where the verification loss parameters are given by  $\theta_{ve}$ .

(Refer Slide Time: 11:43)



## DeepID2: Algorithm

Table 1: The DeepID2 learning algorithm.

**input:** training set  $\chi = \{(x_i, l_i)\}$ , initialized parameters  $\theta_c, \theta_{id}$ , and  $\theta_{ve}$ , hyperparameter  $\lambda$ , learning rate  $\eta(t)$ ,  $t \leftarrow 0$

**while not converge do**

$t \leftarrow t + 1$  sample two training samples  $(x_i, l_i)$  and  $(x_j, l_j)$  from  $\chi$

$f_i = \text{Conv}(x_i, \theta_c)$  and  $f_j = \text{Conv}(x_j, \theta_c)$

$\nabla \theta_{id} = \frac{\partial \text{Ident}(f_i, l_i, \theta_{id})}{\partial \theta_{id}} + \frac{\partial \text{Ident}(f_j, l_j, \theta_{id})}{\partial \theta_{id}}$

$\nabla \theta_{ve} = \lambda \cdot \frac{\partial \text{Verif}(f_i, f_j, y_{ij}, \theta_{ve})}{\partial \theta_{ve}}$ , where  $y_{ij} = 1$  if  $l_i = l_j$ , and  $y_{ij} = -1$  otherwise.

$\nabla f_i = \frac{\partial \text{Ident}(f_i, l_i, \theta_{id})}{\partial f_i} + \lambda \cdot \frac{\partial \text{Verif}(f_i, f_j, y_{ij}, \theta_{ve})}{\partial f_i}$

$\nabla f_j = \frac{\partial \text{Ident}(f_j, l_j, \theta_{id})}{\partial f_j} + \lambda \cdot \frac{\partial \text{Verif}(f_i, f_j, y_{ij}, \theta_{ve})}{\partial f_j}$

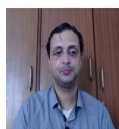
$\nabla \theta_c = \nabla f_i \cdot \frac{\partial \text{Conv}(x_i, \theta_c)}{\partial \theta_c} + \nabla f_j \cdot \frac{\partial \text{Conv}(x_j, \theta_c)}{\partial \theta_c}$

update  $\theta_{id} = \theta_{id} - \eta(t) \cdot \nabla \theta_{id}$ ,  $\theta_{ve} = \theta_{ve} - \eta(t) \cdot \nabla \theta_{ve}$ , and  $\theta_c = \theta_c - \eta(t) \cdot \nabla \theta_c$ .

**end while**

**output**  $\theta_c$

- $\theta_{id}$  and  $\theta_{ve}$ : separately updated via respective objectives;  $\theta_c$  (backbone CNN parameters): trained via weighted contribution from identification and verification objectives
- enables backbone network to extract inter-personal features (via identification signal), and intra-personal features (via verification signal)



Vineeth N B. (IIT-H)

§7.4 CNNs for Human Understanding: Faces

21 / 37

But DeepID2 uses a different way of learning the entire pipeline, which is slightly different. And let us see this algorithm here. So, given that you sample 2 training samples  $(x_i, l_i)$  and  $(x_j, l_j)$  from your training data set. So, you now have  $f_i$ , which is the output of the convolution layer for  $x_i$  and  $f_j$ , which is the output of the basic backbone CNN for  $x_j$ . These, this backbone CNN is parametrized by weights  $\theta_c$ . Now, the gradient of  $\theta_{id}$  is given by  $\frac{\partial Ident(f_i, l_i, \theta_{id})}{\partial \theta_{id}} + \frac{\partial Ident(f_j, l_j, \theta_{id})}{\partial \theta_{id}}$ . So, in this case, you are using the cross-entropy loss for both  $f_i$  and  $f_j$ .

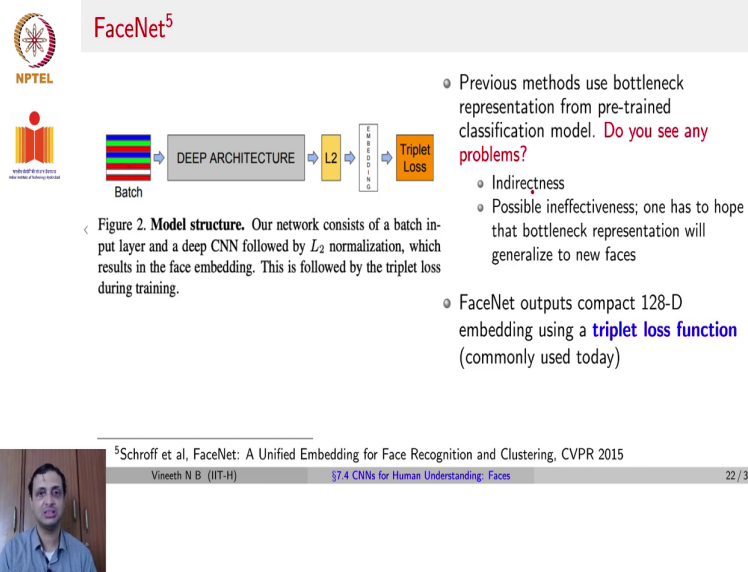
Similarly, the gradient for the verification parameters is given by  $\lambda \cdot \frac{\partial Verif(f_i, f_j, y_{ij}, \theta_{ve})}{\partial \theta_{ve}}$ , where the verification loss is given by the pairwise contrastive loss, where we know that in this particular case  $y_{ij}$  was set to 1 if the identities were the same and  $y_{ij}$  was set to minus 1 if the identities were dissimilar. Then similarly,  $\nabla f_i$  and  $\nabla f_j$  was given by, remember  $f_i$  and  $f_j$  are the (param) the gradient with respect to the embeddings  $f_i$  and  $f_j$ , which are outputs of the CNN, which is given by  $\partial Ident + \partial Verif$  for  $f_i$  and  $\nabla f_j$ , similarly for  $f_j$ .

And the final gradient for the entire CNNs parameters is given by  $\nabla f_i \frac{\partial Conv(x_i, \theta_c)}{\partial \theta_c} + \nabla f_j \frac{\partial Conv(x_j, \theta_c)}{\partial \theta_c}$ . Rather, you are ensuring that the weight updates for the CNN are shared between the gradient, are firstly updated using the gradients using both the  $x_i$  outputs and the  $x_j$  outputs, and the same weights are then shared for both the CNN architectures that are applied on both  $x_i$  and  $x_j$ .

So, to summarize,  $\theta_{id}$  and  $\theta_{ve}$  are separately updated via respective objectives. But  $\theta_c$ , which are the weights of the backbone CNN before you branch out into verification and identification are trained via a weighted contribution from both identification and verification objectives. This enables the backbone network to both learn good inter-personal features and intra-personal features.



(Refer Slide Time: 15:09)



The diagram shows the FaceNet architecture: a 'Batch' of input images goes into a 'DEEP ARCHITECTURE' (represented by a stack of colored rectangles), followed by an 'L2' normalization step (a yellow box), and finally a 'Triplet Loss' step (an orange box). The output is a '128-D' embedding (a vertical stack of boxes).

**FaceNet<sup>5</sup>**

NPTEL  
National Programme on Technology Enhanced Learning

◦ Previous methods use bottleneck representation from pre-trained classification model. **Do you see any problems?**

- Indirectness
- Possible ineffectiveness; one has to hope that bottleneck representation will generalize to new faces

◦ FaceNet outputs compact 128-D embedding using a **triplet loss function** (commonly used today)


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by  $L_2$  normalization, which results in the face embedding. This is followed by the triplet loss during training.

<sup>5</sup>Schroff et al, FaceNet: A Unified Embedding for Face Recognition and Clustering, CVPR 2015  
Vineeth N. B. (IIT-H) 57.4 CNNs for Human Understanding: Faces 22 / 37

Subsequently, came FaceNet, which was developed in CVPR of 2015, which premised that existing architectures for face recognition, including DeepFace and DeepID2 typically learned an entire neural network, and then used one of the penultimate layers, which is called a bottleneck layer for, and those embeddings for any final decision making in matching. Is there a problem with this approach? FaceNet claimed, yes, it has an indirectness, because we do not know what those embeddings from an intermediate layer or a penultimate layer or what is called as a bottleneck layer could be used for in later tasks.

And because it was not explicitly trained for a later face or later task that those embeddings are used for, it may be ineffective. So, FaceNet rather tries to ensure that its goal is to output a good representation. There are no further goals of classification, which are considered separately based on the learned representation. In fact, FaceNet learns a 128-dimensional embedding using a new loss function known as a Triplet Loss Function, which is commonly used across several settings in deep learning today.

(Refer Slide Time: 16:37)



### FaceNet: Triplet Loss





Figure 3. The **Triplet Loss** minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity.

- Alternate formulation of contrastive loss; considers triplet of inputs (namely anchor, positive, negative) instead of input pairs
- **Goal:** Ensure distance between anchor ( $x_a$ ) and negative sample ( $x_n$ ) representation is at least "margin" more than distance between anchor and positive sample ( $x_p$ )
- Triplet Constraint:
 
$$\|f(x_a) - f(x_p)\|_2^2 + \alpha < \|f(x_a) - f(x_n)\|_2^2$$

where  $\alpha$  is margin





Vineeth N B. (IIT-H)      §7.4 CNNs for Human Understanding: Faces      23 / 37

So, here is the idea of triplet loss. It is an extension of contrastive loss. While contrastive loss used similar and dissimilar examples, triplet loss uses a triplet. What is the triplet made of? An anchor point, a positive point and a negative point. So, given any input, which is an anchor point for us now, we now then have a positive point, which has the same identity as the anchor, and the negative point, which has a different identity as the anchor. We now try to use all these 3 in, while learning through a loss function. And our goal here, obviously, is to ensure that the distance between the anchor and the negative sample is at least a certain margin more than the distance between the anchor and the positive sample.

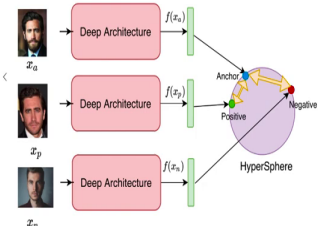
How do we implement this using a loss function? We have a triplet constraint that states  $\|f(x_a) - f(x_p)\|_2^2 + \alpha < \|f(x_a) - f(x_n)\|_2^2$ . We are saying that the distance of the anchor to the negative sample must be at least a certain margin  $m$ , which is  $m$  is a positive, alpha here, alpha is a positive quantity more than the distance of the anchor to  $x_p$ , which is the positive sample.


(Refer Slide Time: 18:13)

### FaceNet: Triplet Loss Formulation

- Let  $f(x)$ : representation/embedding on d-dimensional hypersphere s.t.  $\|f(x)\|_2 = 1$
- **Goal:** train  $\theta$  to ensure that for all triplets  $x_a$ (anchor),  $x_p$ (positive),  $x_n$ (negative):
 
$$\|f(x_a) - f(x_p)\|_2^2 + \alpha < \|f(x_a) - f(x_n)\|_2^2$$
 where  $\alpha$  is margin
- Achieved by training parameters  $\theta$  to minimize:
 
$$L_{triplet} = \sum_i [\|f(x_a^i) - f(x_p^i)\|_2^2 - \|f(x_a^i) - f(x_n^i)\|_2^2 + \alpha]$$





Vineeth N B. (IIT-H) §7.4 CNNs for Human Understanding: Faces 24 / 37



So, you could visualize this as, you have an anchor image that is given as input to your CNN model. You have a positive example for that anchor image and you also have a negative example for that anchor image in your data set. You first ensure that after you get the representations as output of your CNN architecture, the same CNN architecture is used for all these 3 inputs, very similar to before. You ensure that these representations are normalized, so that you get  $\|f(x)\|_2 = 1$  for each of these representations. So, you normalize those vector representations that you get.

Now, this would be equivalent to each of these points lying on a unit hypersphere or a unit ball in whichever dimensional space, 128-dimensional space that you are considering here. And your goal now is to train  $\theta$  or the weights of the neural network, given all these triplets to ensure that  $\|f(x_a) - f(x_p)\|_2^2 + \alpha < \|f(x_a) - f(x_n)\|_2^2$ , where alpha is margin. This is achieved by using the triplet loss, which is given by

$$\|f(x_a) - f(x_p)\|_2^2 - \|f(x_a) - f(x_n)\|_2^2 + \alpha$$


It is simple to see that this will now try to ensure that this loss at least has a certain margin of alpha between the distance to the negative sample to the anchor and the positive sample to the anchor.

(Refer Slide Time: 19:55)

### Triplet Selection Strategy

- How to choose triplets?
- “Easy” triplets can:
  - inhibit learning; model does not strive to learn parameters capturing distinctive features
  - have slower convergence
- Important to select triplets that violate triplet constraint initially, enabling the model to “work hard” to satisfy the constraint



Vineeth N B. (IIT-H)     §7.4 CNNs for Human Understanding: Faces     25 / 37

One question here is, “How do you select these triplets, given an anchor image?” You could have what are known as Easy Triplets, where its very clear as to what the positive sample is and what the negative sample is. However, easy triplets may not really facilitate learning, they may not be able to allow the network to learn distinctive features. So, it may be important to select triplets that initially violate the triplet constraint, so that the network learns. And eventually, perhaps over time change the nature of triplets.

(Refer Slide Time: 20:39)




### Hard Triplet Mining

- Selecting “hard” triplets  $\implies$  choose  $x_p^i$  (hard positive):  $\arg \max_{x_p^i} \|f(x_a^i) - f(x_p^i)\|_2^2$   
 $x_n^i$  (hard negative):  $\arg \min_{x_n^i} \|f(x_a^i) - f(x_n^i)\|_2^2$
- FaceNet uses online triplet sampling; hard positive/negatives sampled from every mini-batch
- **Problem:** Very hard negatives early on  $\implies$  bad local minima initially, potential training/model collapse
- **Solution:** “Semi-hard” negatives  $\implies$  negatives that lie inside margin  $\alpha$ ; how? Choose  $x_n^i$  (semi-hard):  $\|f(x_a) - f(x_p)\|_2^2 < \|f(x_a) - f(x_n)\|_2^2$



Vineeth N B. (IIT-H)     §7.4 CNNs for Human Understanding: Faces     26 / 37

So, you could first choose what are known as Hard Triplets. Given a mini-batch of triplet samples that you are going to use (the) to train this neural network, you could choose only the

hard positives and the hard negatives. What are hard positives? Where  $f(x_a) - f(x_p)$  is maximized. Those samples, where the distance to positive samples is the highest, would be a hard positive. And where the distance of the anchor to a negative is minimum, would be a hard negative. So, we choose those kinds of samples in a given mini-batch as hard triplets that we can train the network on.

So, FaceNet actually uses online triplet sampling, where hard positives and negatives are sampled from every mini-batch. There is one problem here though, if you choose very hard negatives early on, there could be bad local minima, and the training could collapse. So, the idea would be to choose semi-hard negatives in a initial stage, and then move on to more harder negatives in later stage of training. How do you choose semi-hard negatives? We ideally want to choose negatives that lie inside the margin between, alpha was the margin, so we want to choose those kinds of samples to start with. It may not necessarily be the minimum distance, but we want to choose a negative sample that is inside the margin.

How do you do? How do you choose such a semi-hard example? You can pick  $\|f(x_a) - f(x_p)\|_2^2 < \|f(x_a) - f(x_n)\|_2^2$ . You ignore the  $\alpha$  that we had, and this way, you may end up choosing samples that are lying inside the margin while you train initially.

(Refer Slide Time: 22:39)

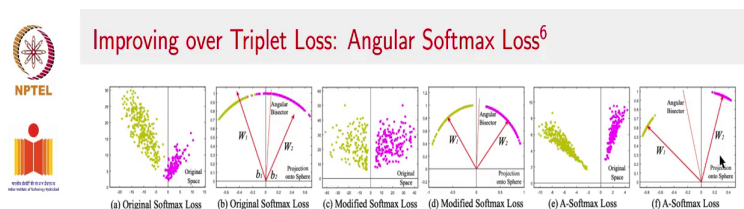
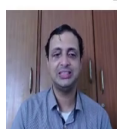


Figure 2: Comparison among softmax loss, modified softmax loss and A-Softmax loss. In this toy experiment, we construct a CNN to learn 2-D features on a subset of the CASIA face dataset. In specific, we set the output dimension of FC1 layer as 2 and visualize the learned features. Yellow dots represent the first class face features, while purple dots represent the second class face features. One can see that features learned by the original softmax loss can not be classified simply via angles, while modified softmax loss can. Our A-Softmax loss can further increase the angular margin of learned features.

- Features learned by softmax loss have intrinsic angular distribution  $\implies$  Euclidean margin-based losses incompatible with softmax loss
- **A-Softmax loss** hence uses angle between feature and classification layer vector, both in training and test



<sup>6</sup>Liu et al, SphereFace: Deep Hypersphere Embedding for Face Recognition, CVPR 2017


Going further over triplet loss, in 2017, there was an approach known as Angular Softmax Loss that intended to improve identification and verification performance with faces. It can also be used for other domains. And these set of diagrams from the paper illustrate the overall

idea. So, if you had an original softmax loss, and you considered the embeddings of the neural network before applying the softmax loss, whatever embedding you had from the penultimate layer, before applying the Softmax Activation function and the cross-entropy loss after that, you may have an embedding something like this, where all these points, yellow in colour belong to one class, all these points that are pink in colour or purple in colour belong to the other class.

So, one can see that if you now embedded, if you normalize these vectors and get a unit norm, it may be a little difficult to use the cosine distance between these two to separate the identities. You can see here that there are certain points, where the yellow and the purple almost overlap. This could be improved by using a Modified Softmax Loss, where you normalize your weight vectors before obtaining these embeddings. We will see the formal definition soon.

And this particular work proposes an Angular Softmax, where you enhance the distinction between the positive class embeddings and the negative class embeddings, so that then you could use a cosine distance metric to effectively separate the identities or get similarity between the identities. Why is this important in this context? Remember that for most of these identification or verification methods, the last step is a step of matching using a distance metric. And one distance metric that could be used for matching is the cosine similarity or the angle between these representations. So, we now look at different definitions of these modified Softmax losses.

(Refer Slide Time: 24:59)




### A-Softmax Loss

- **Softmax Loss:**

$$\mathcal{L}_{softmax} = \frac{1}{N} \sum_i -\log \left( \frac{e^{W_i^T x_i + b_i}}{\sum_j e^{W_j^T x_i + b_j}} \right) = \frac{1}{N} \sum_i -\log \left( \frac{e^{\|W_i\| \|x_i\| \cos(\theta_{y_i, i}) + b_i}}{\sum_j e^{\|W_j\| \|x_i\| \cos(\theta_{j, i}) + b_j}} \right)$$
- **Modified Softmax Loss** ( $\|W_1\| = \|W_2\| = 1, b_1 = b_2 = 0$ ):
 
$$\mathcal{L}_{modified} = \frac{1}{N} \sum_i -\log \left( \frac{e^{\|x_i\| \cos(\theta_{y_i, i})}}{\sum_j e^{\|x_i\| \cos(\theta_{j, i})}} \right)$$

$\cos \theta_i > \cos \theta_j$   
 $\uparrow$   
 $\cos(m\theta_i)$
- **A-Softmax Loss:**

$$\mathcal{L}_{ang} = \frac{1}{N} \sum_i -\log \left( \frac{e^{\|x_i\| \cos(\alpha \theta_{y_i, i})}}{e^{\|x_i\| \cos(m \theta_{y_i, i})} + \sum_{j \neq y_i} e^{\|x_i\| \cos(\theta_{j, i})}} \right)$$



Vineeth N B. (IIT-H)
§7.4 CNNs for Human Understanding: Faces
28 / 37


So, the original Softmax loss can be written as e power, ideally, this would have been  $Z_i$ , which would have been the logits of the neural network. But those logits can be written as  $Wx_i + b$ , where  $x_i$ s are the activations of the layer before applying the Softmax. So, that is what you have as the definition of Softmax. Now, one could write out this dot product  $Wx$  that you have here, as  $\|W\| \|x_i\| \cos(\theta_{y_i}) + b_{y_i}$ , that is the definition of the dot product, plus the bias. Remember, we are looking at the  $y_i$ th label, and that is why this subscript  $b_{y_i}$ . So, this is your standard Softmax loss.

The modified Softmax loss is given by, after your penultimate layer of the neural network, before you apply the Softmax, you normalize the weights that you obtain, and then apply the Softmax. So, in this case, the norms of the weights  $W_1$  and  $W_2$  for any other neuron, remember that you are going to have different weights for every neuron connecting to that last layer, so each of those weights are normalized to 1. Now, your Softmax loss becomes this first term here,  $\|W\|$  becomes 1.

And it also ensures that the bias becomes 0. That is also done as part of the operation. So, your new modified loss looks something like this, where the  $\|W\|$  becomes 1 and the bias becomes 0. These are operations that are performed before applying the Softmax. The angular Softmax loss extend this, extends this to make the angle corresponding to the positive class be scaled by  $m$ , while the angles corresponding to all other classes not scaled by the same  $m$  value.  $m$  is a positive constant. This ensures that the angle is further separated between the positive class and the rest of the classes.


Remember, that the final match to check whether a feature belongs to one class and the other class would be given by where  $\cos \theta_i > \cos \theta_j$ , you would then say that this point belongs to the  $i$ th label and not the  $j$ th label. And by doing Angular Softmax, we are ensuring that we would now have  $\cos(m\theta_i)$ , which is likely to be greater than  $\cos \theta_j$ , we are trying to ensure that for the positive class, the cost value can be further separated from the cost value for the angle with respect to the other classes. That is the main idea for the Angular Softmax.

(Refer Slide Time: 28:03)

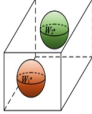


### A-Softmax Loss: Binary Classification Analysis

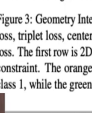
  



Euclidean Margin Loss



Modified Softmax Loss




A-Softmax Loss ( $m \geq 2$ )

Figure 3: Geometry Interpretation of Euclidean margin loss (e.g. contrastive loss, triplet loss, center loss, etc.), modified softmax loss and A-Softmax loss. The first row is 2D feature constraint, and the second row is 3D feature constraint. The orange region indicates the discriminative constraint for class 1, while the green region is for class 2.

Let  $W_i, b_i$  be weights and bias in softmax loss:

- **Softmax** decision boundary  $\implies (W_1 - W_2)x + b_1 - b_2 = 0$
- Constrain  $\|W_1\| = \|W_2\| = 1, b_1 = b_2 = 0;$   
**Modified Softmax** decision boundary  $\implies \|x\|(\cos(\theta_1) - \cos(\theta_2)) = 0$
- Margin  $m \geq 1$ : quantitatively controls the size of angular margin  
**A-Softmax** decision boundary  $\implies$   
 class 1:  $\|x\|(\cos(m\theta_1) - \cos(\theta_2)) = 0$   
 class 2:  $\|x\|(\cos(\theta_1) - \cos(m\theta_2)) = 0$



Vineeth N B. (IIT-H) §7.4 CNNs for Human Understanding: Faces 29 / 37



So, here is another visualization from the same paper. The Softmax decision boundary can be given by  $(W_1 - W_2)x + b_1 - b_2$ . Taking the activations from 2 different classes, remember, you are going to have multiple classes in that last layer. We are considering 2 classes,  $W_1$ , and the weights corresponding to them are  $W_1$  and  $W_2$ . And this is your Softmax decision boundary. You see here, that when you have your Softmax decision boundary, you are separating those 2 classes, this particular way.

In your modified Softmax boundary, you are normalizing your weights, and hence, all your points lie on a unit ball. And you are now trying to ensure that you get a separation such as this. Your points belong to positive classes lie on one side of the ball and your points belonging to the negative class lie on the other side of the ball. This is a little bit better, because in the previous Softmax approach, you do not know where these 2 balls may overlap with each other. You are now trying to ensure that these go on different sides of the same ball.

But this still have a problem with modified soft max, at these points, which are close to each other at where the ball, where the classes meet in the ball. And angular Softmax tries to ensure that that separation is further maximized by having  $W_1$ s in a further arc on one side of the ball and  $W_2$ s on the other side of the arc in the other side of the ball. The further separation helps better classification in the final step.



(Refer Slide Time: 29:57)

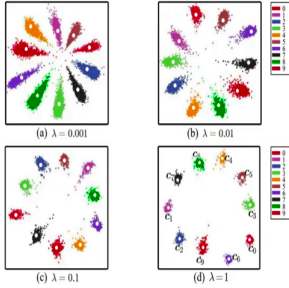
  


### Other Loss Functions used: CenterLoss<sup>7</sup>

- Intended to augment Softmax+CE to also reduce intra-class variance

$$\mathcal{L}_{new} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{CL}$$
$$= \mathcal{L}_{CE} + \frac{\lambda}{2} \sum_{i=1}^m \|x_i - c_{y_i}\|_2^2$$

- Added term minimizes intra-class distance between sample  $x$  and  $y_i$ th class centroid  $c_{y_i}$  of deep features
- Centroid is updated online during learning





<sup>7</sup>Wen et al, Discriminative Feature Learning Approach for Deep Face Recognition, ECCV 2016  
Vineeth N B. (IIT-H) 37.4 CNNs for Human Understanding: Faces 30 / 37

Over the last few years, there have been other ideas to improve these loss functions. A lot of these have been used in the face analysis context. CenterLoss is another approach that was proposed in 2016, which was a loss added to the cross-entropy loss, where in addition to the standard cross-entropy loss, we also obtain a centroid for each class label. And for a point belonging to a particular class, we also minimize the distance of this representation to the centroid of that particular class, which this data point belongs to. So, these centroids are also updated online during learning as you keep training over different iterations.

You get, you get these centroids based on the representations in each particular iteration over training. And in that iteration, we try to ensure that the representations are close to the centroid of that particular class. And the diagram on the left, on the right shows that as you vary the coefficient  $\lambda$ , which weights, how, which tells you how much you are going to weigh that distance to the centroid. You can see that initially when  $\lambda$  has a very small value, you get a certain separation with respect to the centroid.

And as you wait,  $\lambda$  more and more, you see that all the embeddings of a particular class end up going close to the centroid, which are given by these white dots in each of these classes. This allows good separation between the classes, when you want to make your final decision on whether there is a match with a certain identity or a match versus a no match.


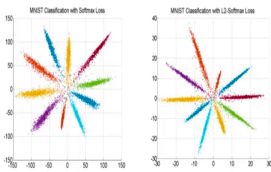
(Refer Slide Time: 31:54)


  


### L2-Softmax<sup>8</sup>

- Enforces L2-norm of features to be fixed for every face image by adding an L2 constraint to feature descriptor such that it lies on hypersphere of fixed radius ( $\alpha$ )
- Features with **high L2 norm** are **easy** to classify for network (spatially placed at boundaries in feature space)
- Features with **low L2 norm** are **difficult** to classify (spatially located near origin)

minimize  $\mathcal{L}_{CE}$   
subject to  $\|f(x_i)\|_2 = \alpha, i = 1, \dots, m.$




<sup>8</sup>Ranjan et al, L2-constrained Softmax Loss for Discriminative Face Verification, 2017  
Vineeth N B. (IIT-H) [§7.4 CNNs for Human Understanding: Faces](#) 31 / 37

L2-Softmax is another improvement, which tries to ensure that the L2-norm of the features before the Softmax lie on a certain hypersphere of a fixed radius  $\lambda$ . Instead of normalizing the features to ensure they have a unit norm, this approach tries to ensure that they have a particular norm  $\lambda$ . And this, the intuition behind this approach comes from their observation that features with high L2-norm are easy to classify and features with low L2-norm because they could be close to the origin could be more difficult to classify.


When it is more close to the origin, the points could look more close to each other. When the points have a high L2-norm, they are further away from the origin and hence could be separated a bit more easily. So, in this case, the optimization turns out to be, you minimize cross-entropy loss such that the representations that you get before you apply a Softmax ensures that  $\|f(x_i)\|_2 = \alpha$ .

(Refer Slide Time: 33:14)

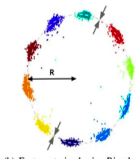


### RingLoss<sup>9</sup>

- While L2-Softmax normalizes features with a *projection* method, **RingLoss** enforces feature normalizations directly in loss function; radius R is learned in training process



$$\begin{aligned} & \underset{x}{\text{minimize}} && \mathcal{L}_{CE} \\ & \text{subject to} && \|f(x_i)\|_2 = R \end{aligned}$$



- Implemented as:


$$\mathcal{L}_R = \frac{\lambda}{2m} \sum_{i=1}^m (\|f(x_i)\|_2 - R)^2$$

<sup>9</sup>Zheng et al, Ring Loss: Convex Feature Normalization for Face Recognition, CVPR 2018

Vineeth N. B. (IIT-H) [§7.4 CNNs for Human Understanding: Faces](#) 32 / 37

This was further improved in RingLoss in 2018, which is this very similar idea as L2-Softmax. However, even in this case, you minimize the cross-entropy loss such that  $\|f(x_i)\|_2 = R$ . But the difference from L2-Softmax is that in this particular case, you also learn the R as part of your training process. So, the neural network decides what should be the value of the L2-norm value also in addition to learning the weights of the neural network. This is known as RingLoss, because you are trying to project the feature representations before the Softmax layer onto a ring, whose radius is also decided by the neural network.

(Refer Slide Time: 34:06)



### Other Recent Efforts

- CosFace**<sup>10</sup>: proposes a large margin-based cosine loss (LMCL) for face recognition
- UniformFace**<sup>11</sup>: proposes uniform loss to learn equidistributed representations to fully exploit feature space
- RegularFace**<sup>12</sup>: proposes exclusive regularization to explicitly enlarge angular distance between different identities
- GroupFace**<sup>13</sup>: proposes to utilize multiple group-aware representations, simultaneously, to improve quality of embedding
- CurricularFace**<sup>14</sup>: proposes Adaptive Curriculum Learning loss to adaptively adjust relative importance of easy and hard samples during different training stages

<sup>10</sup>Wang et al, CosFace: Large Margin Cosine Loss for Deep Face Recognition, CVPR 2018

<sup>11</sup>Duan et al, UniformFace: Learning Deep Equidistributed Representation for Face Recognition, CVPR 2019

<sup>12</sup>Zhao et al, RegularFace: Deep Face Recognition via Exclusive Regularization, CVPR 2019

<sup>13</sup>Kim et al, GroupFace: Learning Latent Groups and Constructing Group-based Representations for Face Recognition, CVPR 2020

<sup>14</sup>Huang et al, CurricularFace: Adaptive Curriculum Learning Loss for Deep Face Recognition, CVPR 2020

Vineeth N. B. (IIT-H) [§7.4 CNNs for Human Understanding: Faces](#) 33 / 37

Based on these approaches, there have been a slew of efforts. CosFace proposed in 2018 proposes a large margin-based cosine loss for face recognition. UniformFace proposed in 2019, proposes a uniform loss to learn equidistributed representations, so that you exploit the full feature space rather than focus on one ball around the origin. RegularFace proposed in 2019, again proposes an exclusive regularization to explicitly enlarge angular distance between different entities. GroupFace, which was very recently proposed in 2020 uses multiple group-aware representations to improve the quality of the embedding.

CurricularFace, which was again a recent work in 2020, proposes adaptive curriculum learning to adjust the relative importance of easy and hard samples in different training stages. Curriculum learning is a facet of training deep neural networks, where initially you expose the model to say, simple training examples, and then gradually increase the difficulty of the training examples to help the neural network learn better. So, we saw this with triplet mining, where we said, “let us start with semi-hard negatives, before we go to hard negatives.” And this approach called CurricularFace took this further to propose an adaptive curriculum learning loss.

(Refer Slide Time: 35:49)



The slide is titled "Face Recognition: Closed-Set vs Open-Set". It lists the following loss functions:

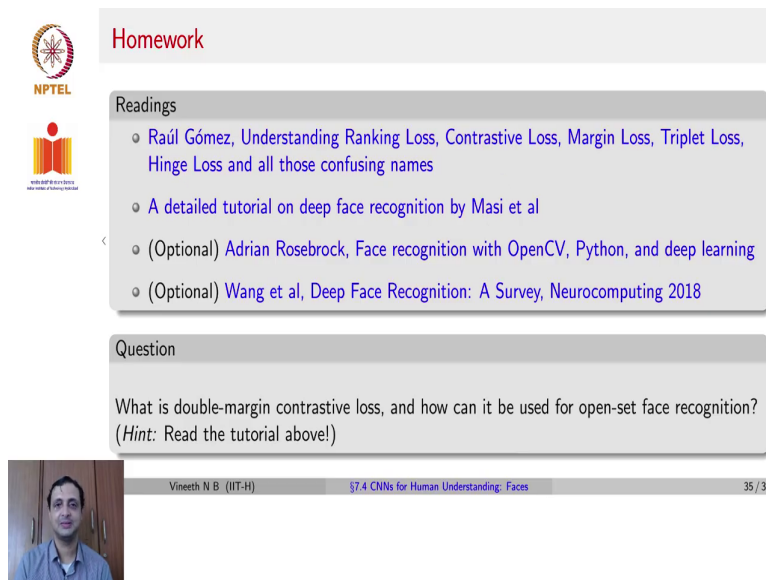
- Closed-Set Loss Functions** (all subjects known)
  - Softmax + CrossEntropy
  - Center Loss (reduce intra-class variability)
  - L2-Softmax (reduce intra-class variability)
  - RingLoss (reduce intra-class variability)
  - Angular Softmax (increase margin between subjects)
- Open-Set Loss Functions** (all subjects not known)
  - Double Margin Contrastive Loss (Homework!)
  - Triplet Loss

Source: Masi et al, Deep Face Recognition: A Survey, SIBGRAP/ 2018  
 Vineth N B. (IIT-H) 37.4 CNNs for Human Understanding: Faces 34 / 37

To conclude this lecture, face recognition also has what is known as a Closed-Set and an Open-Set setting. In a Closed-Set setting, you have a set of pre-given identities that you have to match an input image to. And one could use a Softmax plus CrossEntropy loss or a Center Loss, L2 loss, RingLoss, Angular Softmax Loss, so on and so forth to implement Closed-Set face recognition.

On the other hand, in Open-Set face recognition, one could have people, images of people that are not in your database, and the neural network has to at least say that this person does not belong to the database and is unknown to the current system. How do you, what kind of loss functions do you use to train such a network? There is what is known as Double Margin Contrastive Loss, that could be repurposed to achieve this. Even triplet loss could be used to achieve Open-Set face recognition. For more details, you can look at this work called Deep Face Recognition: A Survey by Masi and others.

(Refer Slide Time: 37:03)



**Homework**

**Readings**

- Raúl Gómez, Understanding Ranking Loss, Contrastive Loss, Margin Loss, Triplet Loss, Hinge Loss and all those confusing names
- A detailed tutorial on deep face recognition by Masi et al
- (Optional) Adrian Rosebrock, Face recognition with OpenCV, Python, and deep learning
- (Optional) Wang et al, Deep Face Recognition: A Survey, Neurocomputing 2018

**Question**

What is double-margin contrastive loss, and how can it be used for open-set face recognition?  
(Hint: Read the tutorial above!)

Vineeth N B (IIT-H) §7.4 CNNs for Human Understanding: Faces 35 / 37

So, your homework readings for this lecture is going to be a very nice blog on understanding Ranking Loss, Contrastive Loss, Margin Loss, Triplet Loss, so on and so forth, in case they are all confusing to you. A very detailed tutorial on deep face recognition by Masi and others. And if you are interested, look at a Python code base for face recognition with OpenCV and also the entire paper on Deep Face Recognition: A Survey. So, the question that we left behind was, “What is Double Margin Contrastive Loss? And how can it be used for Open-Set face recognition?” The hint for you is read this tutorial by Masi et al to get your answer.