In project

MTCNN 🡪 gives coordinates of face so we crop

Apply facenet on it and get embeddings

Apply SVM to classify

MTCNN

MTCNN performs quite fast on a CPU, even though S3FD is still quicker running on a GPU

detector = mtcnn.MTCNN()

faces = detector.detect\_faces(pixels)

Mtcnn output

{'box': [1942, 716, 334, 415], 'confidence': 0.9999997615814209, 'keypoints': {'left\_eye': (2053, 901), 'right\_eye': (2205, 897), 'nose': (2139, 976), 'mouth\_left': (2058, 1029), 'mouth\_right': (2206, 1023)}}  
{'box': [2084, 396, 37, 46], 'confidence': 0.9999206066131592, 'keypoints': {'left\_eye': (2094, 414), 'right\_eye': (2112, 414), 'nose': (2102, 426), 'mouth\_left': (2095, 432), 'mouth\_right': (2112, 431)}}  
{'box': [1980, 381, 44, 59], 'confidence': 0.9998701810836792, 'keypoints': {'left\_eye': (1997, 404), 'right\_eye': (2019, 407), 'nose': (2010, 417), 'mouth\_left': (1995, 425), 'mouth\_right': (2015, 427)}}  
{'box': [2039, 395, 39, 46], 'confidence': 0.9993435740470886, 'keypoints': {'left\_eye': (2054, 409), 'right\_eye': (2071, 415), 'nose': (2058, 422), 'mouth\_left': (2048, 425), 'mouth\_right': (2065, 431)}}

To use MTCNN on a GPU you will need to set up CUDA, cudnn, pytorch

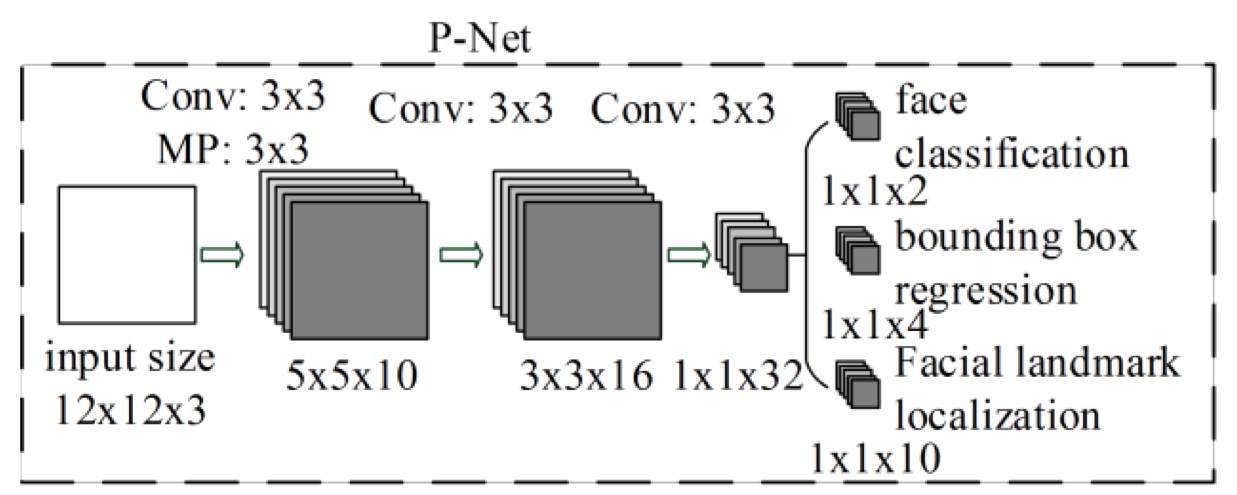
fast\_mtcnn = FastMTCNN(  
stride=4,  
resize=0.5,  
margin=14,  
factor=0.6,  
keep\_all=True,  
device=device  
)

faces = fast\_mtcnn(frames)

Three Stages of MTCNN

## **Stage 1: The Proposal Network (P-Net)**

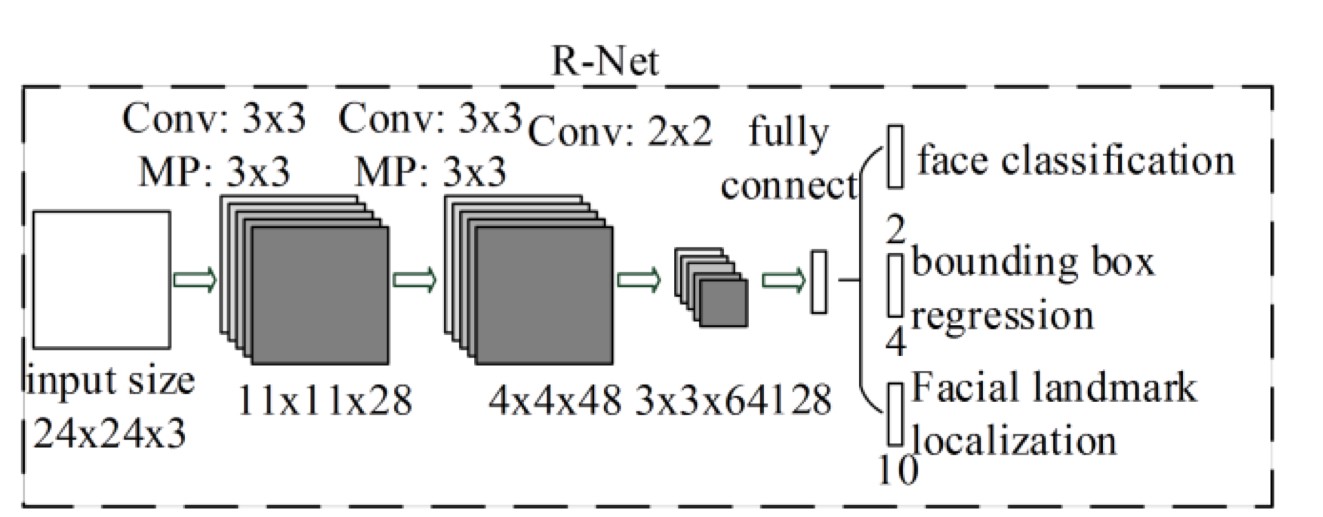
This first stage is a fully convolutional network (FCN). The difference between a CNN and a FCN is that a fully convolutional network does not use a dense layer as part of the architechture. This Proposal Network is used to obtain candidate windows and their bounding box regression vectors.



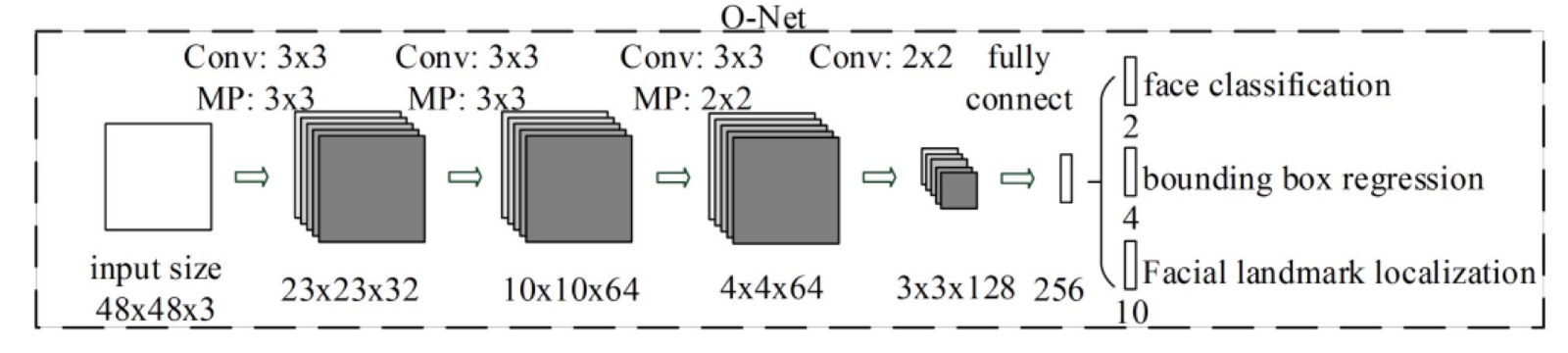
## **Stage 2: The Refine Network (R-Net)**

All candidates from the P-Net are fed into the Refine Network. Notice that this network is a CNN, not a FCN like the one before since there is a dense layer at the last stage of the network architecture. The R-Net further reduces the number of candidates, performs calibration with bounding box regression and employs non-maximum suppression (NMS) to merge overlapping candidates.

The R-Net outputs wether the input is a face or not, a 4 element vector which is the bounding box for the face, and a 10 element vector for facial landmark localization.



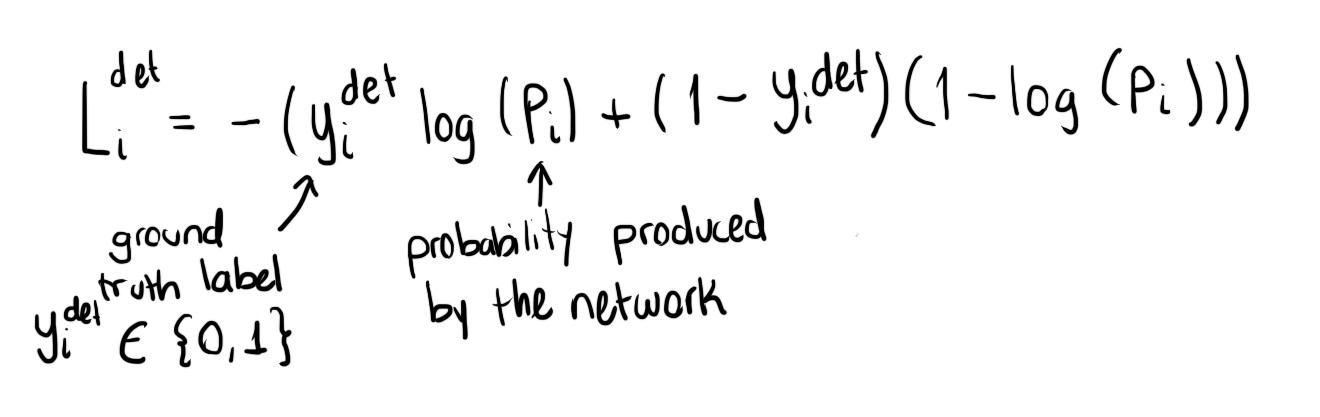
This stage is similar to the R-Net, but this Output Network aims to describe the face in more detail and output the five facial landmarks’ positions for eyes, nose and mouth.



# Three Tasks of MTCNN

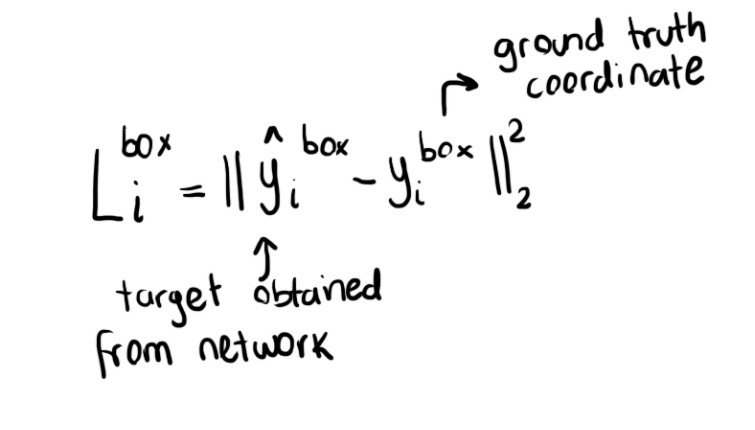
The Network’s task is to output three things: face/non-face classification, bounding box regression, and facial landmark localization.

1. **Face classification**: this is a binary classification problem that uses cross-entropy loss:



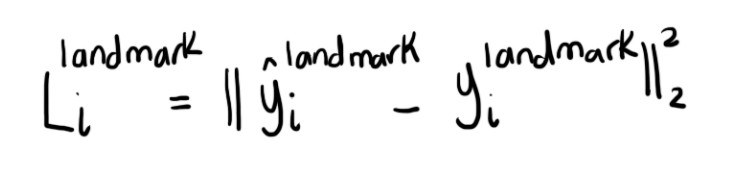
Cross-entropy loss

2. **Bounding box regression**: the learning objective is a regression problem. For each candidate window, the offset between the candidate and the nearest ground truth is calculated. Euclidean loss is employed for this task:



Euclidean loss

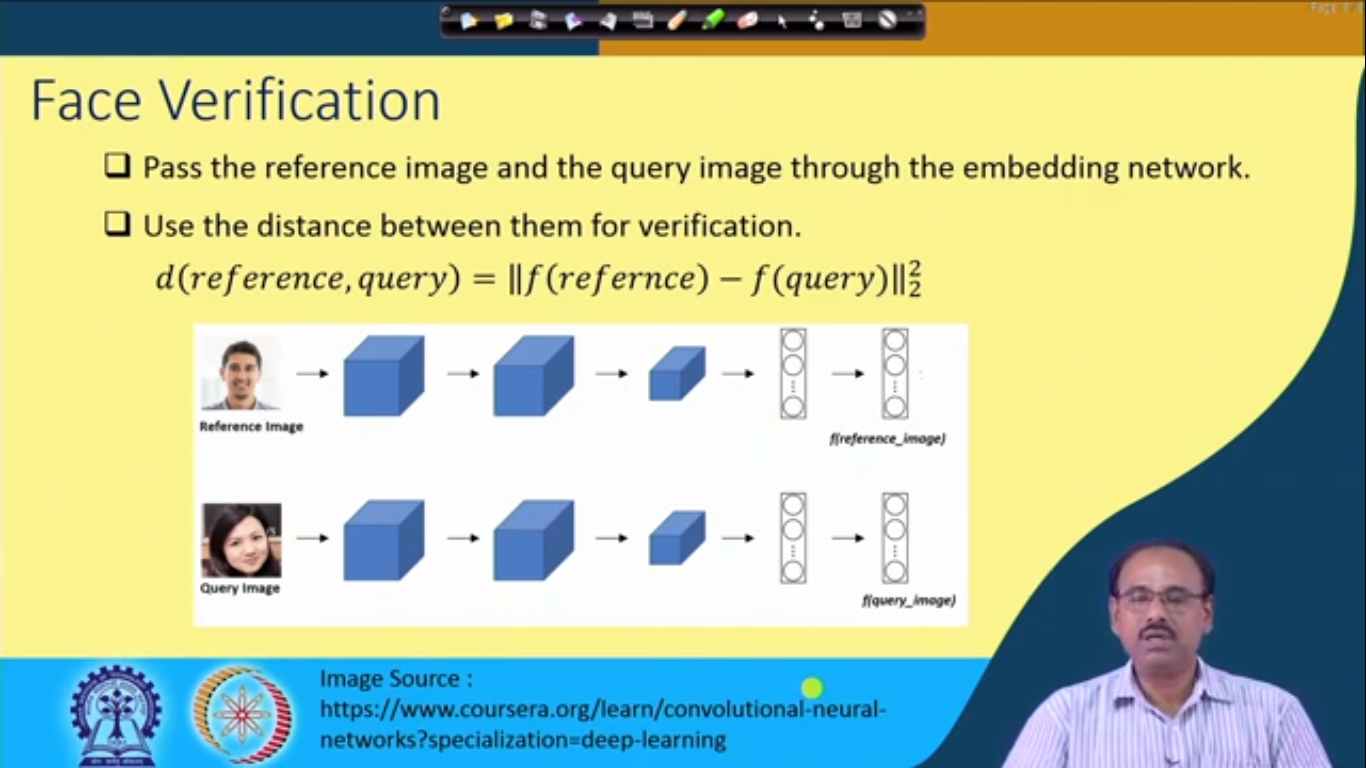
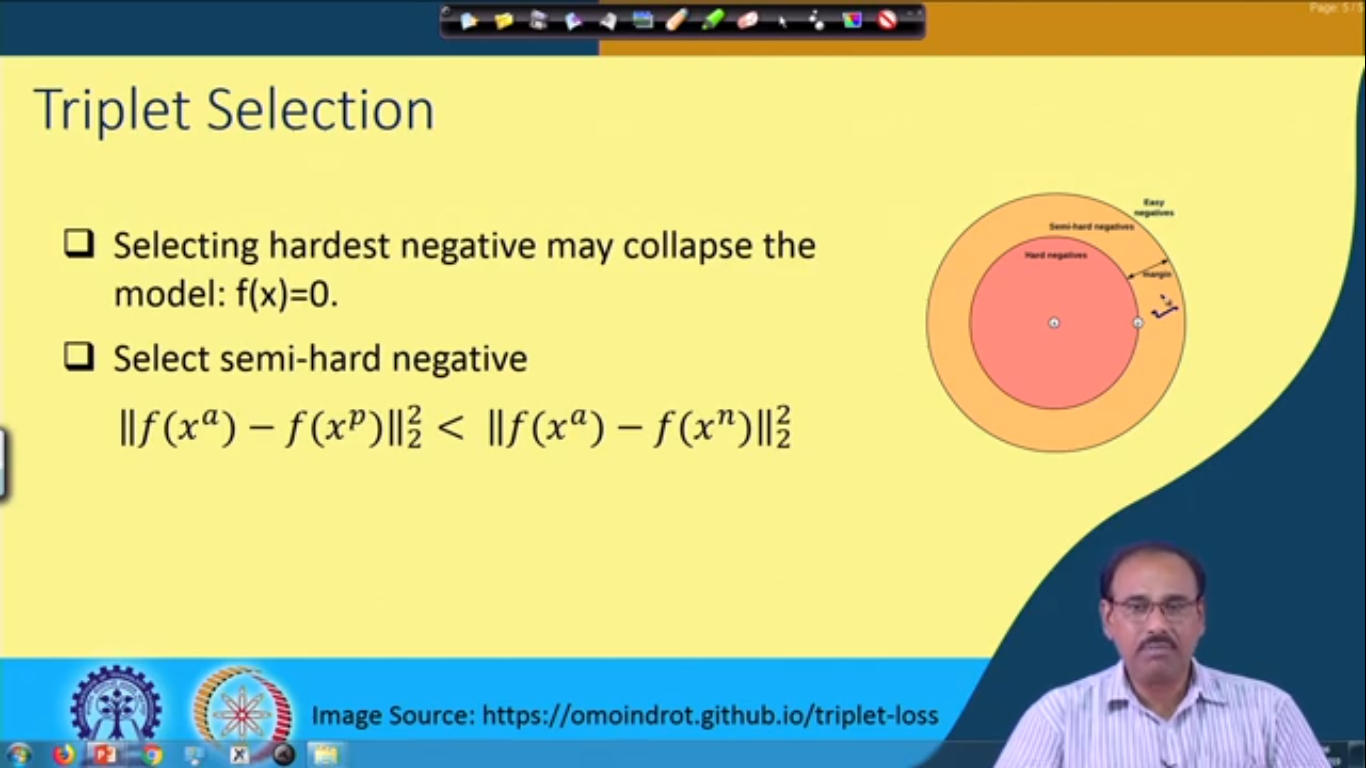
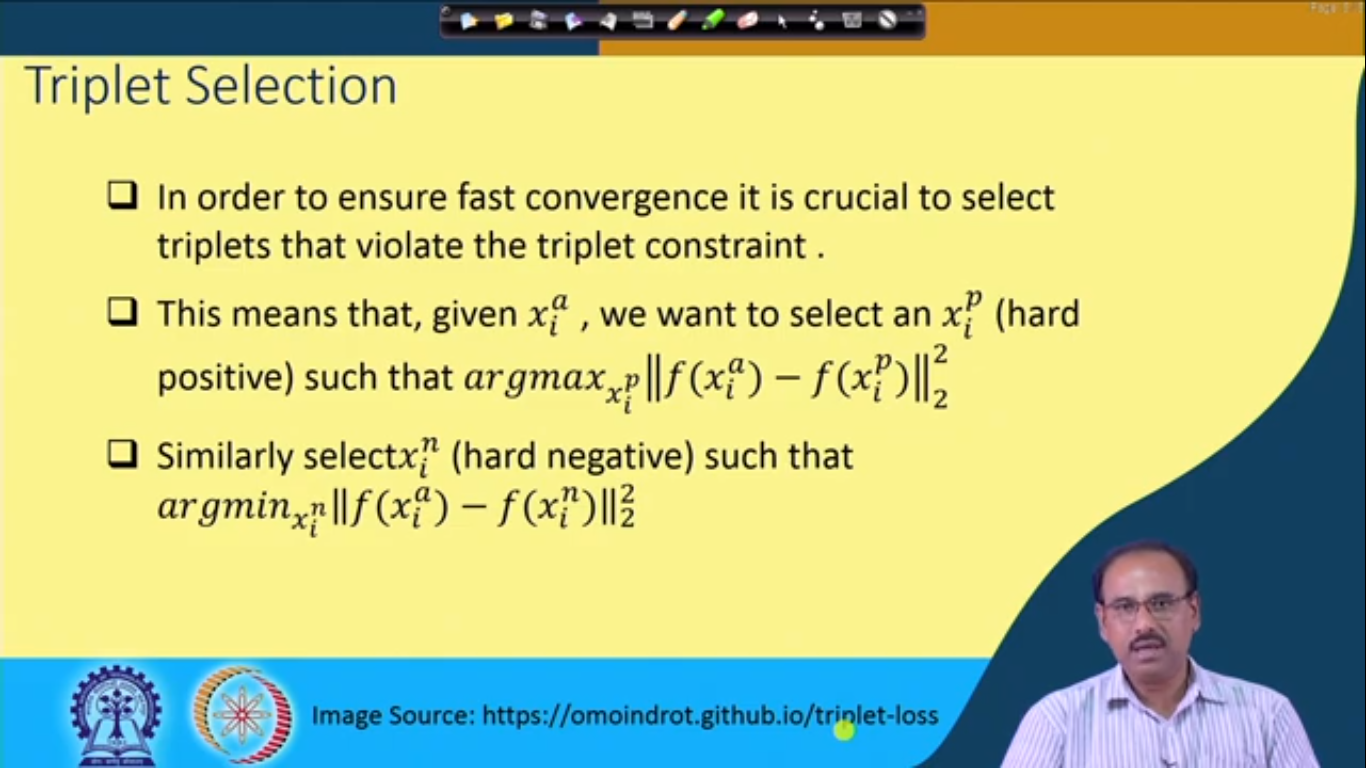
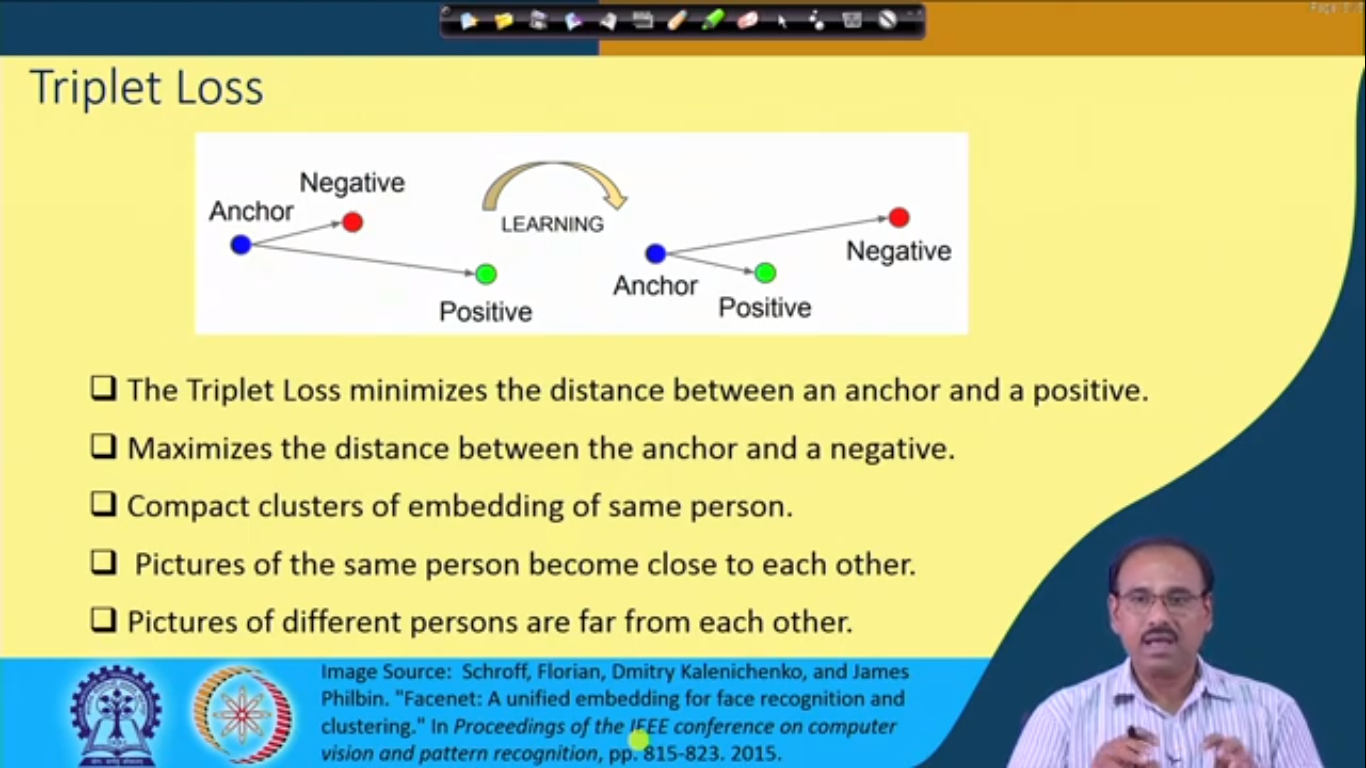
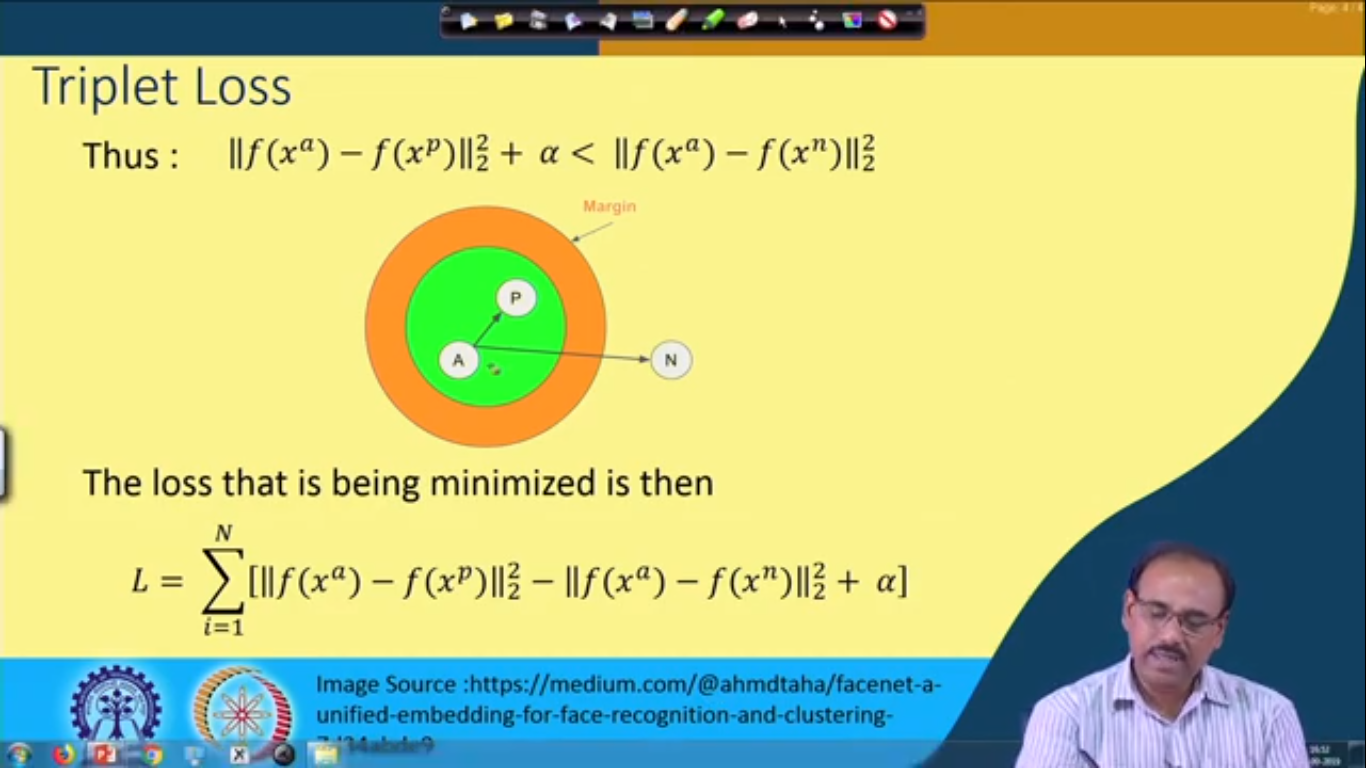
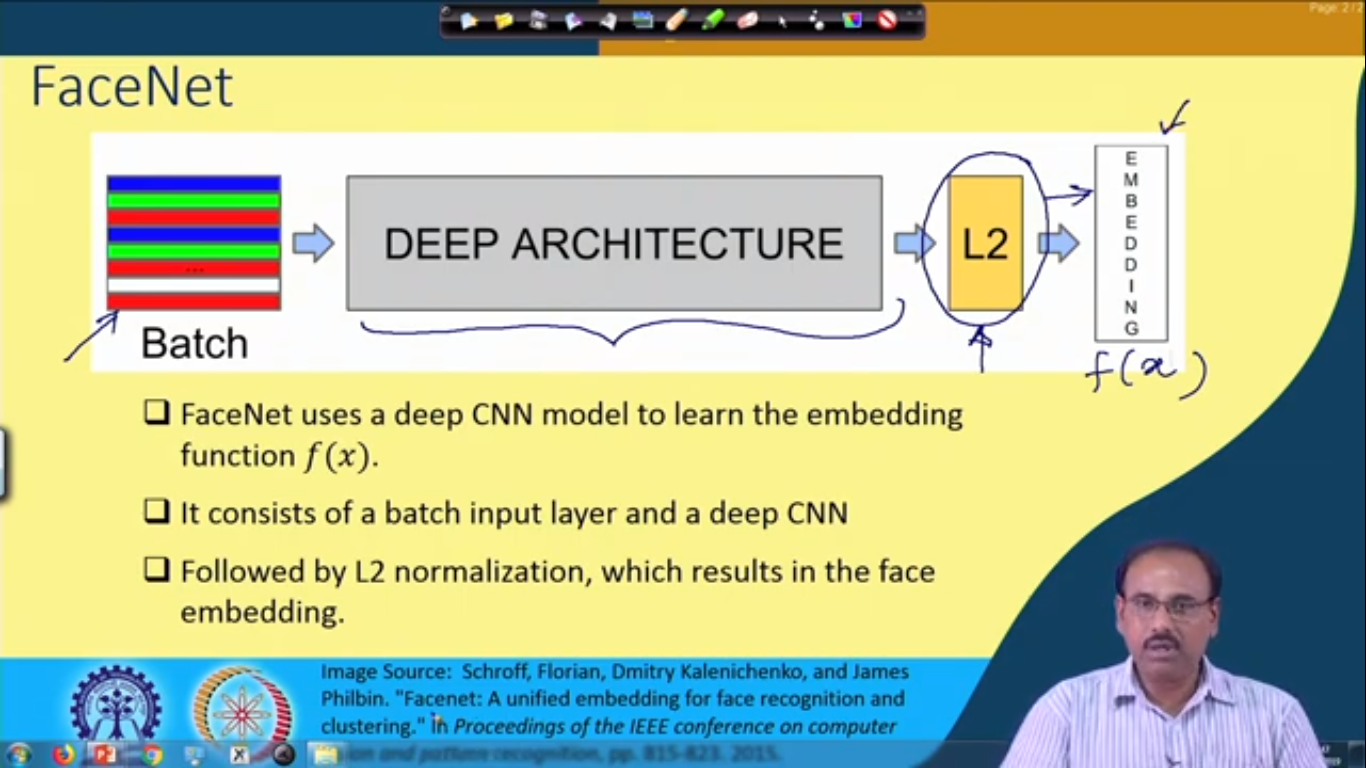
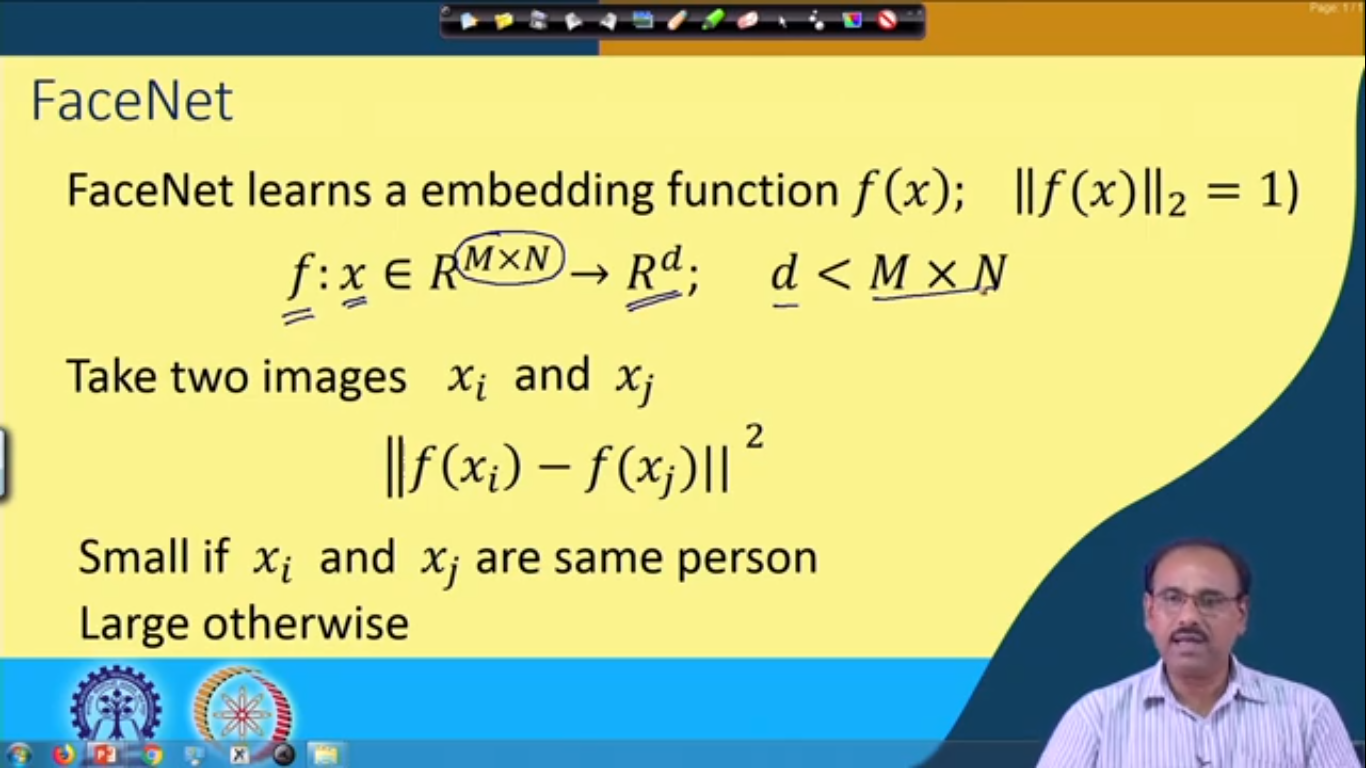
1. **Facial Landmark localization**: the localization of facial landmarks is formulated as a regression problem, in which the loss function is Euclidean distance:



Euclidean loss

There are five landmarks: left eye, right eye, nose, left mouth corner and right mouth corner.

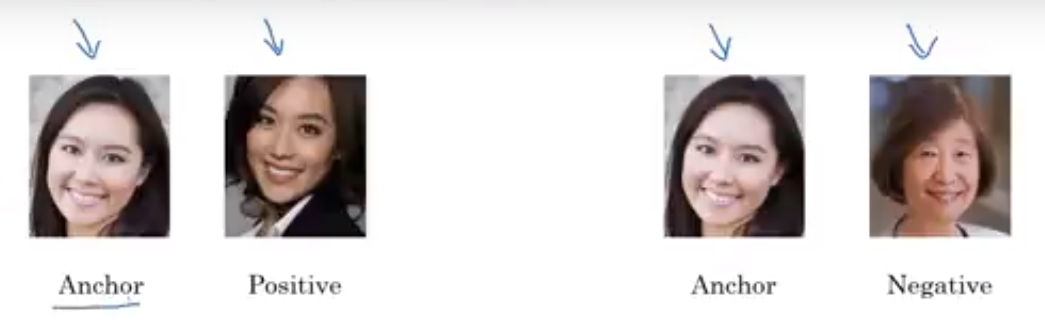
**Facennet**

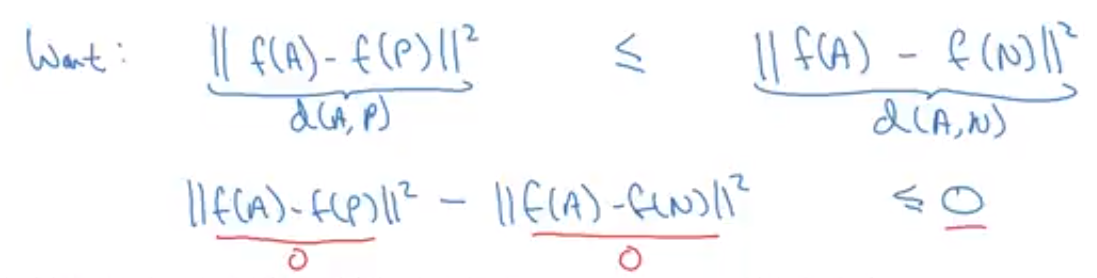


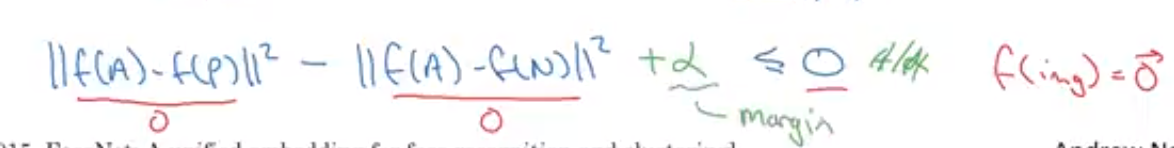
Facenet Triplet loss

(<https://www.youtube.com/watch?v=LN3RdUFPYyI0>)

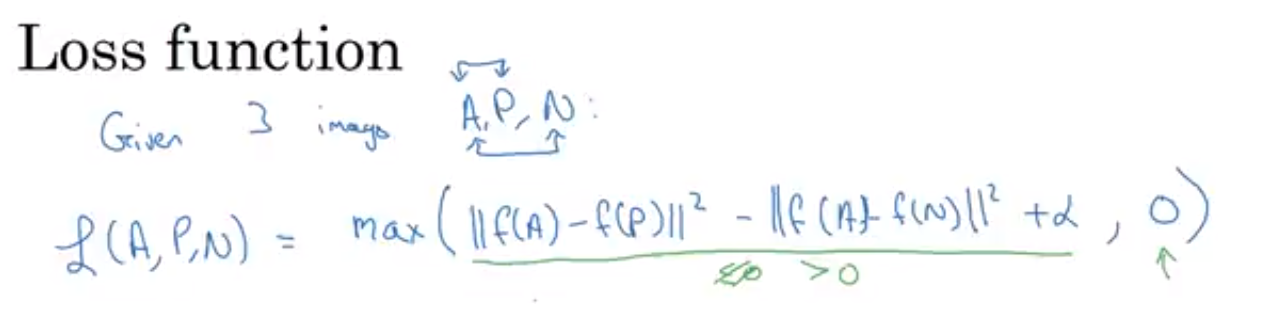
Apply gradient decent on triplet loss

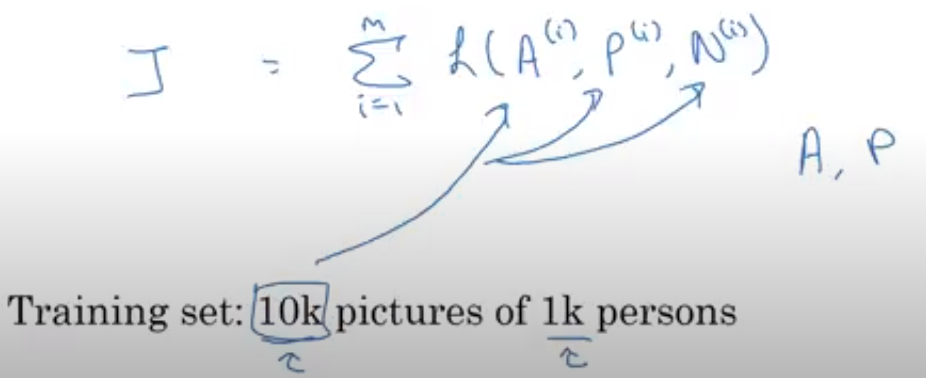






Margin added so that it should not be zero





For training we require multiple pictures of same person

