

# IT499-BS

## Lab 2

# Installation & Required Libraries

```
pip install numpy pandas matplotlib scikit-learn opencv-python pillow scipy  
torch torchvision torchaudio mtcnn facenet-pytorch insightface scikit-image  
scikit-optimize requests
```

# LFW dataset

- Labeled Faces in the Wild - face recognition benchmark
- 13,233 images, 5,749 subjects ( $\sim 1,680$  with  $\geq 2$  images)
- Use **200 subjects with  $\geq 4$  images each**
- Official: <http://vis-www.cs.umass.edu/lfw/lfw.tgz>

```
from sklearn.datasets import fetch_lfw_people
```

# Gallery Set

- Gallery (Enrollment)
- Reference templates of enrolled subjects
- **First 50% of each subject's images**
- **Variable** (typically 2-5 images/subject)
- Used for **1:N matching**

# Probe Set

## Probe (Query)

- Unknown faces to identify
- **Remaining 50% of enrolled subjects' images**
- **Variable | Used to test identification**

# Preprocessing Step 1: Face Detection

- Locate face region in image
- MTCNN detector with confidence filter
- Bounding box: [x, y, width, height]
- Must handle "no face found" cases

```
from mtcnn import MTCNN
```

## Preprocessing Step 2: Face Alignment

- Rotate face to canonical position using landmarks
- Eye-to-eye distance normalized
- Rotation angle computed from eye coordinates
- Improves feature consistency

```
keypoints = {'left_eye', 'right_eye', 'nose', ...}
```

## Preprocessing Step 3: Face Resizing

- Standardize input dimensions
- FaceNet:  $160 \times 160$
- ArcFace:  $112 \times 112$
- Must match model input

```
cv2.resize(face, (160, 160))
```

# Preprocessing Step 4: Normalization

- Map pixel values to model's expected range
- **FaceNet**: ImageNet mean/std
- **ArcFace**: [-1, 1] via 1/127.5 scaling
- Critical for model performance

$\text{img} = (\text{img} - \text{mean}) / \text{std}$

# FEATURE EXTRACTION & MODELS

LBP (Classical) Local Binary Pattern:

- Hand-crafted descriptor, histogram-based
- 256-D
- Grayscale, any size
- LFW (baseline)
- Comparison baseline, interpretability

# FEATURE EXTRACTION & MODELS

## LBP (Classical) Local Binary Pattern:

- LBP Parameters:
  - radius=3, n\_points=24 ( $8 \times \text{radius}$ ), bins=256
  - Creates 256-D histogram
  - Requires grayscale conversion
  - No pre-training needed
- **from skimage.feature import local\_binary\_pattern**

# FEATURE EXTRACTION & MODELS

FaceNet (Deep):

- **InceptionResNetV1 backbone**
- **512-D**
- **160×160 RGB**
- VGGFace2 / CASIA-WebFace
-

# FEATURE EXTRACTION & MODELS

## FaceNet Setup

- Use facenet-pytorch library
- 512-D embedding
- 160×160 + prewhitened
- Pre-trained VGGFace2

```
from facenet_pytorch import MTCNN, InceptionResnetV1
```

# FEATURE EXTRACTION & MODELS

## FaceNet Code

- Load model, preprocess, extract embeddings
- MTCNN → 160×160 → InceptionResnetV1
- L2-normalize embeddings

```
facenet = InceptionResnetV1(pretrained='vggface2')
```

# FEATURE EXTRACTION & MODELS

ArcFace (Deep):

- ResNet-based with angular margin loss
- 512-D
- 112×112 RGB
- MS1M-ArcFace, VGGFace2
- Best-in-class performance

# FEATURE EXTRACTION & MODELS

## ArcFace Setup

- Use insightface library's FaceAnalysis
- 512-D embedding
- 112×112 internally handled
- Pre-trained buffalo\_l model

```
import insightface; app = FaceAnalysis(name='buffalo_l')
```

# FEATURE EXTRACTION & MODELS

ArcFace Code:

- One-line API for detection→alignment→embedding
- Handles all preprocessing internally
- Pass raw BGR image → get 512-D embedding
- L2-normalize for consistency

**emb = app.get(img).embedding**

# MATCHING & METRICS

## Cosine Similarity

- Closed-Set identification
- $\text{cos\_sim} = (\mathbf{x} \cdot \mathbf{y}) / (\|\mathbf{x}\| \|\mathbf{y}\|)$
- [-1, +1], higher is better
- No Threshold (ranking only)

```
from scipy.spatial.distance import cosine; 1 - cosine(v1, v2)
```

# MATCHING & METRICS

## Euclidean Distance

- Open-Set identification
- $d = \|x - y\|_2$
- $[0, \infty]$ , lower is better
- Yes (reject if  $d >$  threshold)

```
from scipy.spatial.distance import euclidean
```

# MATCHING & METRICS

## CMC Curve (Rank-k TPIR)

- **Closed-Set evaluation**
- **Fraction of probes with correct identity in top-k**
- **0–100%**

# MATCHING & METRICS

## FPIR-False Positive ID Rate

- **Open-Set evaluation**
- **Fraction of impostor probes incorrectly accepted**
- **0–1 or 0–100%**
- **Threshold Varies ( $10^{-2}$ ,  $10^{-4}$ ,  $10^{-6}$ )**
- Set threshold to achieve target FPIR

# MATCHING & METRICS

## **FNIR- False Negative ID Rate**

- **Open-Set evaluation**
- **Fraction of genuine probes incorrectly rejected**
- **0–1**
- **Related to FPIR via threshold**
- **1 - TPIR for genuine probes**

# MATCHING & METRICS

## TPIR - True Positive ID Rate

- Both closed & open-set
- Fraction of correct identifications at given FPIR
- 0–100%
- Report @  $\text{FPIR} = 10^{-2}, 10^{-4}, 10^{-6}$
- $\text{TPIR} = 1 - \text{FNIR}$

# MATCHING & METRICS

## TPIR - True Positive ID Rate

- Both closed & open-set
- Fraction of correct identifications at given FPIR
- 0–100%
- Report @  $\text{FPIR} = 10^{-2}, 10^{-4}, 10^{-6}$
- $\text{TPIR} = 1 - \text{FNIR}$

# MATCHING & METRICS

## ROC Curve

- Open-Set evaluation
- Plot TPIR vs FPIR across threshold range
- Threshold-dependent
- Yes (3 different thresholds)
- Use log scale for FPIR, linear for TPIR

# Load FaceNet

```
from facenet_pytorch import MTCNN, InceptionResnetV1
mtcnn = MTCNN(image_size=160, margin=0, device='cuda')
facenet = InceptionResnetV1(pretrained='vggface2').eval()
```

# Preprocess & Extract FaceNet

```
img = Image.open('face.jpg')
img_tensor = mtcnn(img) # detect, align, resize, prewhiten
emb = facenet(img_tensor.unsqueeze(0)).squeeze()
emb = emb / emb.norm() # L2-normalize
```

# Load ArcFace

```
import insightface  
app = insightface.app.FaceAnalysis(name='buffalo_1')  
app.prepare(ctx_id=0, det_size=(640, 640))
```

# Extract ArcFace

Returns faces list; handles all preprocessing

```
img = cv2.imread('face.jpg') # BGR
faces = app.get(img) # auto detect, align, embed
emb = faces.embedding / np.linalg.norm(faces.embedding)
```

- Compute 1:N Cosine Scores
- Compute CMC Curve
- Compute ROC (Open-Set)