Reviewing our Project Description for task "Face Recognition Attendance System Framework": Assessment of the implementation of the two baseline approaches: classification and metric learning for face verification, as well as any additional approaches used.

--------------------------------------------------------------------------------------------------------------

Then, you will review our Jupyter Notebook file (Project.ipynb) and went through step-by-step what we have done their, their output and any reflection was conducted there.

--------------------------------------------------------------------------------------------------------------

After that, you will have to generate a detailed report with each sections explained in details and elaborate as deep as you can about our methods and results as was conducted in the Jupyter Notebook file, and also explaining the code we used to do the task.

--------------------------------------------------------------------------------------------------------------

Project Description:  
1. Introduction  
Face recognition can be classified into two categories: face classification and face verification. Face classification involves assigning a person’s face to the correct face ID, while face verification involves determining whether two face images belong to the same person. Face classification is considered a closed-set problem, where the face IDs are known in advance by the model. In contrast, face verification is an open-set problem, where the model may encounter previously unseen face identities.  
In this project, you will implement face verification using a convolutional neural network (CNN) to design an end-to-end face recognition attendance system for an enterprise.

2. Face Verification  
The input to your system will be a trial — a pair of face images that may or may not belong to the same person. Your goal is to produce a numerical score that quantifies the similarity between the faces in the two images. A simple approach is to flatten each image matrix into a vector and then calculate the Euclidean distance between the two vectors.

3. Getting Started  
In this project, you will explore the following elements to design a face recognition attendance system.

3.1 Face Embedding  
You are not encouraged to directly compute the distance between two image matrices for two main reasons:  
• Flattened images are usually high-dimensional, leading to increased computational costs.  
• The features in the original images are not sufficiently discriminative.  
Therefore, in this project, your task is to train a convolutional neural network (CNN) to extract compact, low-dimensional features that retain the most important information from the image while remaining discriminative. These compact features will be represented as fixed-length vectors, known as face embeddings. Your end-to-end face verification system will function as follows: given two images, each image is passed through the CNN to generate its corresponding face embedding. An appropriate metric is then applied to the embeddings to produce a similarity score.  
There are different approaches you can use to train a convolutional neural network (CNN) to extract discriminative face embeddings. The two common methods are:  
• Metric Learning (Self-Supervised): Metric learning trains a CNN to map face images into an embedding space where similar faces are close together and different faces are far apart. For example, in the triplet loss method, an image called the anchor is compared to a positive sample (same identity) and a negative sample (different identity). The model learns to minimize the distance between the anchor and the positive, while maximizing the distance between the anchor and the negative.  
• Supervised Learning (Classification-Based): Supervised learning treats face recognition as a classification problem, training the CNN to assign each image to a known identity class. A softmax layer is used during training, and once the model learns to classify correctly, the representation from the layer before the softmax is used as the face embedding.  
In this project, you are expected to implement both approaches and compare their performance.

3.2 System Evaluation for Face Verification  
Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) will be used as evaluation metrics to assess model performance.  
A threshold is used to decide whether to accept or reject a pair — that is, a pair is accepted if the similarity score is above the threshold and rejected if it is below. The ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.  
The AUC of the ROC curve represents the probability that the classifier will rank a randomly chosen positive pair (same person) higher than a randomly chosen negative pair (different people), based on their similarity scores.

3.3 Similarity Distance Metric  
You need to conduct some research to select an appropriate distance metric for the face verification task. The two most commonly used distance metrics are cosine similarity and Euclidean distance. Both metrics are capable of achieving state-of-the-art performance. You are encouraged to test and compare the performance of both.

--------------------------------------------------------------------------------------------------------------

Project.ipynb (breakdown into code snippets and their output):  
Path:  
DATASET\_DIR = "/content/drive/My Drive/COS30082/Project/face\_dataset"  
CLASSIFICATION\_DATA = f"{DATASET\_DIR}/classification\_data"  
VERIFICATION\_DATA = f"{DATASET\_DIR}/verification\_data"  
PAIR\_TXT = f"{DATASET\_DIR}/verification\_pairs\_val.txt"

1. Data Preparation:  
from torchvision import datasets, transforms  
from torch.utils.data import DataLoader  
from torchvision.datasets import ImageFolder

Process the image in correct size

transform = transforms.Compose([  
transforms.Resize((160, 160)),  
transforms.ToTensor()  
])

Process data

train\_dataset = ImageFolder(f"{CLASSIFICATION\_DATA}/train\_data", transform=transform)  
val\_dataset = ImageFolder(f"{CLASSIFICATION\_DATA}/val\_data", transform=transform)

Load data

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)  
val\_loader = DataLoader(val\_dataset, batch\_size=32)

2. CNN Backbone (Shared):  
import torch.nn as nn

class FaceCNN(nn.Module):  
def init(self, embed\_dim=128):  
super(FaceCNN, self).init()  
self.conv = nn.Sequential(  
nn.Conv2d(3, 32, 3, 1), nn.ReLU(), nn.MaxPool2d(2),  
nn.Conv2d(32, 64, 3, 1), nn.ReLU(), nn.MaxPool2d(2),  
nn.Conv2d(64, 128, 3, 1), nn.ReLU(), nn.AdaptiveAvgPool2d((1,1))  
)  
self.embed = nn.Linear(128, embed\_dim)

def forward(self, x):

x = self.conv(x).view(x.size(0), -1)

return self.embed(x)

3. Model Building:  
3a. Supervised Training (Classification):  
class Classifier(nn.Module):  
def init(self, base, num\_classes):  
super().init()  
self.base = base  
self.fc = nn.Linear(128, num\_classes)

def forward(self, x):

emb = self.base(x)

return self.fc(emb)

model\_cls = Classifier(FaceCNN(), num\_classes=len(train\_dataset.classes))

3b. Triplet Loss (Metric Learning):  
import torch.nn.functional as F

class TripletNet(nn.Module):  
def init(self, base):  
super().init()  
self.base = base

def forward(self, anchor, positive, negative):

a = self.base(anchor)

p = self.base(positive)

n = self.base(negative)

return a, p, n

def triplet\_loss(anchor, positive, negative, margin=1.0):  
dist\_pos = F.pairwise\_distance(anchor, positive)  
dist\_neg = F.pairwise\_distance(anchor, negative)  
return torch.mean(torch.clamp(dist\_pos - dist\_neg + margin, min=0.0))

3c. Display Model Summary:  
from torchsummary import summary  
import torch

Set GPU/CPU

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

For the supervised model (classifier)

model\_cls = Classifier(FaceCNN(), num\_classes=len(train\_dataset.classes)).to(device)  
summary(model\_cls, input\_size=(3, 160, 160))

Summarize only the embedding network used in TripletNet

model\_triplet = FaceCNN().to(device)  
summary(model\_triplet, input\_size=(3, 160, 160))

Output:

Layer (type) Output Shape Param #

================================================================  
Conv2d-1 [-1, 32, 158, 158] 896  
ReLU-2 [-1, 32, 158, 158] 0  
MaxPool2d-3 [-1, 32, 79, 79] 0  
Conv2d-4 [-1, 64, 77, 77] 18,496  
ReLU-5 [-1, 64, 77, 77] 0  
MaxPool2d-6 [-1, 64, 38, 38] 0  
Conv2d-7 [-1, 128, 36, 36] 73,856  
ReLU-8 [-1, 128, 36, 36] 0  
AdaptiveAvgPool2d-9 [-1, 128, 1, 1] 0  
Linear-10 [-1, 128] 16,512  
FaceCNN-11 [-1, 128] 0  
Linear-12 [-1, 4000] 516,000

Total params: 625,760  
Trainable params: 625,760  
Non-trainable params: 0

Input size (MB): 0.29  
Forward/backward pass size (MB): 22.77  
Params size (MB): 2.39  
Estimated Total Size (MB): 25.45

Layer (type) Output Shape Param #

================================================================  
Conv2d-1 [-1, 32, 158, 158] 896  
ReLU-2 [-1, 32, 158, 158] 0  
MaxPool2d-3 [-1, 32, 79, 79] 0  
Conv2d-4 [-1, 64, 77, 77] 18,496  
ReLU-5 [-1, 64, 77, 77] 0  
MaxPool2d-6 [-1, 64, 38, 38] 0  
Conv2d-7 [-1, 128, 36, 36] 73,856  
ReLU-8 [-1, 128, 36, 36] 0  
AdaptiveAvgPool2d-9 [-1, 128, 1, 1] 0  
Linear-10 [-1, 128] 16,512

Total params: 109,760  
Trainable params: 109,760  
Non-trainable params: 0

Input size (MB): 0.29  
Forward/backward pass size (MB): 22.74  
Params size (MB): 0.42  
Estimated Total Size (MB): 23.45

3d. Reflection/Insights:  
Model Architecture Summary

FaceCNN Embedding Network (Used in Both Approaches)  
| Metric | Value |  
| -------------------- | -------------------- |  
| Total Parameters | 109,760 |  
| Trainable Parameters | 109,760 |  
| Embedding Dimension | 128 |  
| Input Size | 3 × 160 × 160 |  
| Output | 128-D face embedding |

Insight:

* The embedding network is compact, making it efficient for inference and comparison in verification tasks.
* Layers include 3 convolutional blocks followed by an adaptive average pooling and a final linear embedding layer.
* Ideal for metric learning due to its lightweight nature and sufficient representational power.

Classifier Model (FaceCNN + Linear Softmax Head)  
| Metric | Value |  
| --------------------- | ------------------------------------------------ |  
| Total Parameters | 625,760 |  
| Additional Parameters | +516,000 in the final Linear(128 → 4000) layer |  
| Purpose | Identity classification on 4000 classes |  
| Output | Class logits for each identity |

Insight:

* The classifier adds a large linear layer on top of the shared FaceCNN to predict among 4000 identities.
* This is used in the supervised (classification-based) approach.
* The final embedding layer before the classification head can be reused to extract meaningful representations for verification.

4. Embedding Extraction and Plot  
4a. Embedding Extraction  
def get\_embedding(model, img\_tensor, model\_name="Unknown"):  
model.eval()  
with torch.no\_grad():  
# Adds a new dimension the tensor (at index 0)  
input\_tensor = img\_tensor.unsqueeze(0)  
embedding = model(input\_tensor).squeeze()  
return embedding

4b. Embedding Collection  
from sklearn.manifold import TSNE  
import matplotlib.pyplot as plt  
import numpy as np

Collect embeddings into a stack

def collect\_embeddings(model, dataloader, device, model\_name="Unknown", max\_samples=500):  
model.eval()  
embeddings, labels = [], []  
count = 0  
for imgs, lbls in dataloader:  
imgs = imgs.to(device)  
for img, lbl in zip(imgs, lbls):  
with torch.no\_grad():  
emb = model(img.unsqueeze(0)).squeeze().cpu().numpy()  
embeddings.append(emb)  
labels.append(lbl.item())  
count += 1  
if count >= max\_samples:  
break  
if count >= max\_samples:  
break  
return np.array(embeddings), np.array(labels)

4c. Plot Embedding:  
def plot\_embeddings(embeddings, labels, title="Embeddings"):  
tsne = TSNE(n\_components=2, perplexity=30, n\_iter=1000)  
reduced = tsne.fit\_transform(embeddings)  
plt.figure(figsize=(10, 8))  
scatter = plt.scatter(reduced[:, 0], reduced[:, 1], c=labels, cmap='tab10', alpha=0.7)  
legend = plt.legend(\*scatter.legend\_elements(), title="Classes", loc="best", fontsize='small')  
plt.title(title)  
plt.grid(True)  
plt.show()

Example with classifier model

embeddings, labels = collect\_embeddings(model\_cls, val\_loader, device, model\_name="Classifier-CNN")  
plot\_embeddings(embeddings, labels, title="Classifier-CNN Face Embeddings (t-SNE)")

Example with Triplet model

embeddings\_triplet, labels\_triplet = collect\_embeddings(model\_triplet, val\_loader, device, model\_name="Triplet-CNN")  
plot\_embeddings(embeddings\_triplet, labels\_triplet, title="Triplet-CNN Face Embeddings (t-SNE)")

Output:  
The first and second images

5. Face Verification (Simularity Function)  
from sklearn.metrics.pairwise import cosine\_similarity  
from scipy.spatial.distance import euclidean

def compare\_embeddings(emb1, emb2, metric, model\_name="Unknown"):  
if metric == "cosine":  
sim = cosine\_similarity([emb1], [emb2])[0][0]  
elif metric == "euclidean":  
sim = -euclidean(emb1, emb2) # negative for similarity-based scoring  
else:  
raise ValueError(f"Unknown metric: {metric}")  
print(f"[INFO][{model\_name}|{metric}] Similarity score: {sim:.4f}")  
return sim

6. ROC Curve & AUC Evaluation  
from sklearn.metrics import roc\_curve, auc  
import matplotlib.pyplot as plt  
from PIL import Image  
import os

def evaluate\_verification(model, pair\_file, base\_dir, transform, device, metric, model\_name="Unknown"):  
y\_scores, y\_true = [], [] # Lists to store similarity scores and ground truth labels  
print(f"\n========== [START] EVALUATING {model\_name} ==========")  
print(f"[INFO] Reading pair file: {pair\_file}")  
# Load verification pairs (image1, image2, label) from the pair file  
with open(pair\_file, "r") as f:  
lines = f.readlines()  
print(f"[INFO] Total verification pairs: {len(lines)}")  
# Iterate through each pair entry  
for i, line in enumerate(lines):  
img1\_path, img2\_path, label = line.strip().split()  
# Convert relative paths to full paths  
full\_path\_1 = os.path.join(base\_dir, os.path.basename(img1\_path))  
full\_path\_2 = os.path.join(base\_dir, os.path.basename(img2\_path))  
print(f"[READ:{i}] Pair: {img1\_path} vs {img2\_path}, Label: {label}")  
# Load and preprocess both images  
try:  
img1 = transform(Image.open(full\_path\_1).convert("RGB")).to(device)  
img2 = transform(Image.open(full\_path\_2).convert("RGB")).to(device)  
except Exception as e:  
print(f"[ERROR:{i}] Failed to load images: {e}")  
continue # Skip pair if loading fails  
# Extract embeddings from the model  
try:  
emb1 = get\_embedding(model, img1, model\_name).cpu().numpy()  
emb2 = get\_embedding(model, img2, model\_name).cpu().numpy()  
except Exception as e:  
print(f"[ERROR:{i}] Embedding error: {e}")  
continue # Skip pair if model fails  
# Compare the two embeddings to get a similarity score  
try:  
score = compare\_embeddings(emb1, emb2, metric, model\_name)  
y\_scores.append(score)  
y\_true.append(int(label)) # Ground truth: 1 (same person), 0 (different)  
except Exception as e:  
print(f"[ERROR:{i}] Similarity error: {e}")  
# Log progress every 50 pairs  
if i % 50 == 0:  
print(f"[LOG:{i}] Progress: {i}/{len(lines)}")  
# If no scores were successfully computed, abort  
if not y\_scores:  
print("[ERROR] No similarity scores computed.")  
return None  
# Compute ROC curve and AUC from collected scores and labels  
fpr, tpr, \_ = roc\_curve(y\_true, y\_scores) # FPR: False Positive Rate, TPR: True Positive Rate  
roc\_auc = auc(fpr, tpr) # AUC = Area Under the Curve  
print(f"[RESULT][{model\_name}] AUC Score ({metric}): {roc\_auc:.4f}")  
# Plot the ROC Curve  
plt.figure()  
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc\_auc:.2f})')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title(f'ROC - {model\_name} - ({metric})')  
plt.legend(loc='lower right')  
plt.grid()  
plt.show()  
return roc\_auc # Return the numeric AUC score for further comparison

Example Usage:  
evaluate\_verification(  
model=model\_cls.base,  
pair\_file=PAIR\_TXT,  
base\_dir=VERIFICATION\_DATA,  
transform=transform,  
device=device,  
metric="cosine",  
model\_name="Supervised Classifier"  
)

Result:  
ROC Curve & AUC Comparison  
To evaluate the effectiveness of different embedding approaches and similarity metrics, we compute ROC curves and AUC (Area Under the Curve) values for each model. The ROC curve visualizes the tradeoff between true positive rate and false positive rate, while the AUC score quantifies how well the model distinguishes between positive and negative pairs (same vs. different identities).

AUC Results Table

| Model Type | Distance Metric | AUC Score |
| --- | --- | --- |
| Supervised Classifier | Cosine | 0.51 |
| Supervised Classifier | Euclidean | 0.51 |
| Triplet Embedding | Cosine | 0.52 |
| Triplet Embedding | Euclidean | 0.52 |

Insights & Interpretation  
Supervised Classifier:

* Both cosine and Euclidean scores produced an AUC of 0.51, only slightly above random guessing (0.50).
* This indicates that the embeddings extracted from the classification model are not sufficiently discriminative for verification tasks.
* Likely reason: the classification model may have overfitted to class labels without enforcing embedding separability between identities.

Triplet Embedding Model:

* Slightly improved AUC values (0.52) for both cosine and Euclidean metrics.
* Although the improvement is marginal, it demonstrates that triplet loss provides a more discriminative embedding space by design.
* The small gain suggests either limited training data, suboptimal triplet mining, or early stopping before convergence.

Cosine vs. Euclidean Comparison

* Across both models, cosine similarity and Euclidean distance performed almost identically.
* This is expected when embeddings are not explicitly L2-normalized cosine similarity loses advantage without normalized vectors.

Conclusion  
While both methods currently perform close to random, the triplet embedding approach shows slightly better discriminative power. This aligns with the theoretical benefit of metric learning in face verification.

7. Threshold-based Verification Accuracy  
from sklearn.metrics import roc\_curve, auc  
import matplotlib.pyplot as plt  
from PIL import Image  
import os

def threshold\_verification(model, pair\_file, base\_dir, transform, device, metric, model\_name="Unknown"):  
y\_scores, y\_true = [], [] # Lists to store similarity scores and ground truth labels  
print(f"\n========== [START] EVALUATING {model\_name} ==========")  
print(f"[INFO] Reading pair file: {pair\_file}")  
# Load verification pairs (image1, image2, label) from the pair file  
with open(pair\_file, "r") as f:  
lines = f.readlines()  
print(f"[INFO] Total verification pairs: {len(lines)}")  
# Iterate through each pair entry  
for i, line in enumerate(lines):  
img1\_path, img2\_path, label = line.strip().split()  
# Convert relative paths to full paths  
full\_path\_1 = os.path.join(base\_dir, os.path.basename(img1\_path))  
full\_path\_2 = os.path.join(base\_dir, os.path.basename(img2\_path))  
print(f"[READ:{i}] Pair: {img1\_path} vs {img2\_path}, Label: {label}")  
# Load and preprocess both images  
try:  
img1 = transform(Image.open(full\_path\_1).convert("RGB")).to(device)  
img2 = transform(Image.open(full\_path\_2).convert("RGB")).to(device)  
except Exception as e:  
print(f"[ERROR:{i}] Failed to load images: {e}")  
continue # Skip pair if loading fails  
# Extract embeddings from the model  
try:  
emb1 = get\_embedding(model, img1, model\_name).cpu().numpy()  
emb2 = get\_embedding(model, img2, model\_name).cpu().numpy()  
except Exception as e:  
print(f"[ERROR:{i}] Embedding error: {e}")  
continue # Skip pair if model fails  
# Compare the two embeddings to get a similarity score  
try:  
score = compare\_embeddings(emb1, emb2, metric, model\_name)  
y\_scores.append(score)  
y\_true.append(int(label)) # Ground truth: 1 (same person), 0 (different)  
except Exception as e:  
print(f"[ERROR:{i}] Similarity error: {e}")  
# Log progress every 50 pairs  
if i % 50 == 0:  
print(f"[LOG:{i}] Progress: {i}/{len(lines)}")  
# If no scores were successfully computed, abort  
if not y\_scores:  
print("[ERROR] No similarity scores computed.")  
return None  
# Compute ROC curve and AUC from collected scores and labels  
fpr, tpr, thresholds = roc\_curve(y\_true, y\_scores) # FPR: False Positive Rate, TPR: True Positive Rate + Threshold  
roc\_auc = auc(fpr, tpr) # AUC = Area Under the Curve  
print(f"[RESULT][{model\_name}] AUC Score ({metric}): {roc\_auc:.4f}")  
# Plot the ROC Curve  
plt.figure()  
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc\_auc:.2f})')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title(f'ROC - {model\_name} - ({metric})')  
plt.legend(loc='lower right')  
plt.grid()  
plt.show()

# This compliment to Step 6: Compute accuracy at optimal threshold (Youden's J statistic)

best\_idx = np.argmax(tpr - fpr)

best\_thresh = thresholds[best\_idx]

print(f"[THRESHOLD] Optimal threshold: {best\_thresh:.4f}")

# Apply threshold to get binary predictions

if metric == "cosine":

binary\_preds = [1 if s > best\_thresh else 0 for s in y\_scores]

else:

binary\_preds = [1 if s < best\_thresh else 0 for s in y\_scores] # Lower distance = more similar

# Compute accuracy

acc = np.mean(np.array(binary\_preds) == np.array(y\_true))

print(f"[RESULT][{model\_name}] Verification Accuracy at optimal threshold ({metric}): {acc:.4f}")

return roc\_auc, acc

Output:  
[THRESHOLD] Optimal threshold: 0.9985  
[RESULT][Supervised Classifier] Verification Accuracy at optimal threshold (cosine): 0.5149  
== SUMMARY ==  
AUC Score: 0.5054  
Verification Accuracy at Optimal Threshold: 0.5149

[THRESHOLD] Optimal threshold: -0.0975  
[RESULT][Supervised Classifier] Verification Accuracy at optimal threshold (euclidean): 0.4901  
== SUMMARY ==  
AUC Score: 0.5114  
Verification Accuracy at Optimal Threshold: 0.4901

[THRESHOLD] Optimal threshold: 0.9981  
[RESULT][Triplet Embedding] Verification Accuracy at optimal threshold (cosine): 0.5233  
== SUMMARY ==  
AUC Score: 0.5188  
Verification Accuracy at Optimal Threshold: 0.5233

[THRESHOLD] Optimal threshold: -0.0429  
[RESULT][Triplet Embedding] Verification Accuracy at optimal threshold (euclidean): 0.4762  
== SUMMARY ==  
AUC Score: 0.5226  
Verification Accuracy at Optimal Threshold: 0.4762

Reflection:  
Overview  
In addition to ROC-AUC analysis (Step 6), we further evaluated model performance using a threshold-based decision mechanism to simulate a real-world verification system — where a similarity score between two faces must be accepted or rejected based on a fixed threshold.  
The following script was used to:

* Compute ROC and AUC
* Determine the optimal threshold using Youden's J statistic (i.e., maximizing TPR - FPR)
* Evaluate final verification accuracy at that threshold

Evaluation Summary  
| Model Type | Distance Metric | AUC Score | Accuracy - Optimal Threshold |  
| --------------------- | --------------- | --------- | ---------------------------- |  
| Supervised Classifier | Cosine | 0.5054 | 0.5149 |  
| Supervised Classifier | Euclidean | 0.5114 | 0.4901 |  
| Triplet Embedding | Cosine | 0.5188 | 0.5233 |  
| Triplet Embedding | Euclidean | 0.5226 | 0.4762

Insights & Interpretation

* AUC scores are all near 0.5, indicating random guessing performance for all four settings.
* Verification accuracy is similarly clustered around 50%, indicating the models struggle to reliably separate same vs. different identities based on embedding similarity.
* Triplet embedding with cosine performed marginally best, but the gap is minimal and not statistically significant.

Why the Performance May Be Low

* Undertrained Models: If classification and metric learning models were trained for few epochs or with limited data, the embeddings might not have learned strong facial representations.
* Low Image Resolution or Quality: Preprocessing inconsistencies or poor input quality can lead to noisy embeddings.
* Dataset Challenge: The verification pairs may include highly similar negatives or challenging lighting/pose variations, making it harder for shallow models to distinguish.
* Metric Learning Limitations: Triplet loss requires careful mining (e.g., hard or semi-hard negatives). Random triplets may not provide strong gradients for learning.

Takeaways:

* Threshold-based evaluation complements AUC by simulating a deployable decision system.
* However, AUC > 0.8 is typically required for face verification systems to be considered practically useful.
* These results suggest a need to:
  + Improve CNN architecture (e.g., deeper layers or pretrained backbones like MobileFaceNet)
  + Use smarter sampling strategies for triplet loss
  + Tune hyperparameters (e.g., margin, learning rate, batch size)