

Performance Evaluation of InsightFace Models on LFW, CALFW, and CPLFW Datasets in a Client-Server Framework

Amirhossein Safari

Introduction

The `face_recognition_eval` project, which is discussed in this report, is implemented based on the [InsightFace](#) package. This package is based on the paper "[ArcFace: Additive Angular Margin Loss for Deep Face Recognition](#)" by Jiankang Deng and colleagues. The paper introduces a novel method for optimizing face recognition models using **Additive Angular Margin Loss** (ArcFace), which helps improve the discriminative power of facial features in the hyperspherical space.

Background and Objective

Face recognition, as one of the most important research areas in computer vision, is of high significance, especially in security and identity verification fields. With the increasing volume of data and model complexity, designing appropriate loss functions for training face recognition models becomes challenging. ArcFace, by adding an angular margin to the loss function, helps models position facial features more compactly within classes and more distantly between classes.

Project Structure

The `face_recognition_eval` project utilizes a client-server architecture to evaluate face recognition models using standard datasets like LFW, CALFW, and CPLFW. This project loads and evaluates advanced face recognition models using the InsightFace package and presents results based on various metrics such as accuracy, false match rate, and false non-match rate.

In the following sections of this report, further details about the project architecture, implementation, and results from model evaluations will be discussed.

Implementation Overview

repository implements a client-server architecture using FastAPI for the server and Python for the client. The system is designed to evaluate face recognition models by comparing their performance on standard datasets like LFW, CALFW, and CPLFW.

Server Implementation

Below is a detailed explanation of the implementation and integration of the `face_rec` module with the FastAPI server (`main.py`):

1. The `face_rec` Module (`face_rec.py`)

Purpose:

This module implements the **FaceMatcher** class, which is responsible for detecting faces, extracting facial embeddings, and comparing them using cosine similarity. The main steps involved are:

- **Model Initialization and Preparation:**
 - **Constructor (`__init__`):**

The class is initialized with parameters such as the model name (for example, "buffalo_s" or "buffalo_l"), the context ID (to determine GPU/CPU usage), and the detection size.

It immediately calls the `prepare_model` method.
 - **`prepare_model`:**

This method creates an instance of the InsightFace's `FaceAnalysis` model using the provided model name. It then prepares the model with the specified device context (`ctx_id`) and detection size (`det_size`).

```
1 def __init__(self, model_name: str, ctx_id: int = 0, det_size: tuple = (480, 480)):
2     """
3     Initialize the face analysis model.
4
5     Args:
6         model_name (str): Model name (e.g., "buffalo_s" or "buffalo_l").
7         ctx_id (int, optional): The context ID (GPU/CPU). Defaults to 0.
8         det_size (tuple, optional): Detection size. Defaults to (480, 480).
9     """
10    self.model_name = model_name
11    self.ctx_id = ctx_id
12    self.det_size = det_size
13    self.model = None
14    self.prepare_model()
15
16    def prepare_model(self) -> None:
17        """Prepare the face analysis model."""
18        self.model = FaceAnalysis(name=self.model_name)
19        self.model.prepare(ctx_id=self.ctx_id, det_size=self.det_size)
20
```

- **Image Preprocessing:**

- **preprocess_image:**

This static method takes an image in the form of a byte buffer and converts it to a NumPy array.

- It decodes the image using OpenCV.
- Checks if the decoding was successful (returns `None` for an invalid image).
- Converts the image from BGR to RGB and casts it to `float32`.

```
1 @staticmethod
2 def preprocess_image(image: np.ndarray) -> Optional[np.ndarray]:
3     """
4     Preprocess the input image by decoding and converting it from BGR to RGB.
5
6     Args:
7         image (np.ndarray): The input image as a byte buffer.
8
9     Returns:
10        Optional[np.ndarray]: The preprocessed image in RGB format, or None if decoding fails.
11    """
12    image = np.frombuffer(image, np.uint8)
13    image = cv2.imdecode(image, cv2.IMREAD_COLOR)
14    if image is None:
15        return None # Handle invalid images
16    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB).astype(np.float32)
```

- **Embedding Extraction:**

- **get_face_embedding:**

This method first preprocesses the image and then uses the InsightFace model to detect faces in it.

- If one or more faces are detected, it extracts and returns the embedding (feature vector) of the first face.
- If no face is found or the image fails to decode, it raises a `ValueError` or returns `None`.

```
1 def get_face_embedding(self, image: np.ndarray) -> Optional[np.ndarray]:
2     """
3     Extract the face embedding from an image.
4
5     Args:
6         image (np.ndarray): The input image as a byte buffer.
7
8     Returns:
9         Optional[np.ndarray]: The face embedding, or None if no face is detected.
10
11     Raises:
12         ValueError: If no face is detected in the image.
13    """
14    image = self.preprocess_image(image)
15    if image is None:
16        raise ValueError("Invalid image format or decoding failure.")
17
18    faces = self.model.get(image)
19    return faces[0].embedding if faces else None
```

- **Cosine Similarity Computation:**

- **cosine_similarity:**

A static method that calculates the cosine similarity between two face embeddings.

- It uses the dot product divided by the product of the norms of the two embeddings.

```
1  @staticmethod
2  def cosine_similarity(embedding1: np.ndarray, embedding2: np.ndarray) -> float:
3      """
4      Compute the cosine similarity between two face embeddings.
5
6      Args:
7          embedding1 (np.ndarray): The first face embedding.
8          embedding2 (np.ndarray): The second face embedding.
9
10     Returns:
11         float: Cosine similarity score.
12     """
13     return np.dot(embedding1, embedding2) / (np.linalg.norm(embedding1) * np.linalg.norm(embedding2))
14
```

- **Face Matching:**

- **match_face:**

This method compares two images by:

- Extracting embeddings for both images using `get_face_embedding`.
 - Handling cases where face extraction fails (returning `None`).
 - Calculating the cosine similarity and comparing it with a threshold (default value is 0.22).
 - Returning `True` if the similarity exceeds the threshold, otherwise `False`.

```
1  def match_face(self, image1: np.ndarray, image2: np.ndarray, threshold: float = 0.22) -> Optional[bool]:
2      """
3      Compare two faces and determine if they match.
4
5      Args:
6          image1 (np.ndarray): The first image as a byte buffer.
7          image2 (np.ndarray): The second image as a byte buffer.
8          threshold (float, optional): Similarity threshold. Defaults to 0.25.
9
10     Returns:
11         Optional[bool]: True if the similarity is above the threshold, False otherwise, or None if faces are missing.
12     """
13     try:
14         embedding1 = self.get_face_embedding(image1)
15         embedding2 = self.get_face_embedding(image2)
16     except ValueError:
17         return None # Handle cases where no faces are detected
18
19     if embedding1 is None or embedding2 is None:
20         return None # Handle cases where face detection fails
21
22     return self.cosine_similarity(embedding1, embedding2) > threshold
```

2. FastAPI Server Integration (main.py)

Purpose:

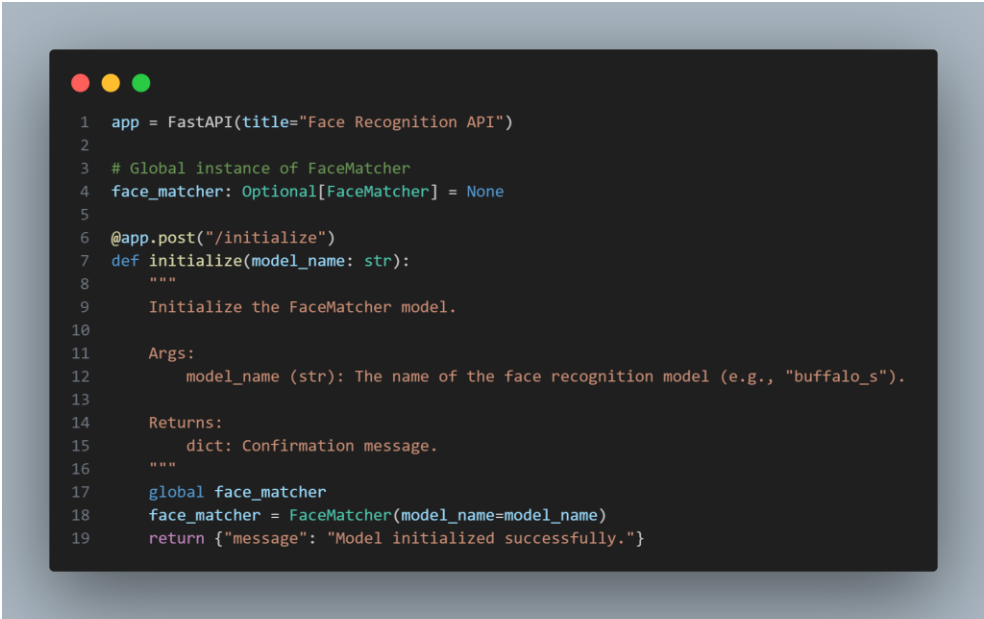
The FastAPI server integrates the **FaceMatcher** class to provide an API for initializing the model and predicting whether two face images match.

- **Global FaceMatcher Instance:**

A global variable `face_matcher` is used to store the instance of **FaceMatcher**. This allows the model to be initialized once and used across multiple API requests.

- **Endpoint /initialize:**

- **Function:** Initializes the FaceMatcher model.
- **Input:** Receives a model name (e.g., "buffalo_s") as a query parameter.
- **Operation:**
Creates a new instance of **FaceMatcher** with the provided model name and assigns it to the global variable.
- **Output:**
Returns a JSON message confirming the successful initialization of the model.



```
1 app = FastAPI(title="Face Recognition API")
2
3 # Global instance of FaceMatcher
4 face_matcher: Optional[FaceMatcher] = None
5
6 @app.post("/initialize")
7 def initialize(model_name: str):
8     """
9     Initialize the FaceMatcher model.
10
11     Args:
12         model_name (str): The name of the face recognition model (e.g., "buffalo_s").
13
14     Returns:
15         dict: Confirmation message.
16     """
17     global face_matcher
18     face_matcher = FaceMatcher(model_name=model_name)
19     return {"message": "Model initialized successfully."}
```

- **Endpoint /predict:**

- **Function:** Compares two uploaded images to determine if they match.
- **Input:**
Receives two files (`file1` and `file2`) as part of the request.
- **Operation:**
 - Reads the content of the uploaded files as byte buffers.
 - Calls `face_matcher.match_face` to compare the two images.
 - Handles errors such as:
 - Uninitialized model (by checking if `face_matcher` is `None`).
 - Cases where no face is detected in one or both images.

- **Output:**
Returns a JSON response with "result": 1 if the faces match (i.e., the cosine similarity is above the threshold), or "result": 0 if they do not. In cases where no face is detected or an error occurs, an appropriate HTTP error is raised.

```

1  @app.post("/predict")
2  async def predict(file1: UploadFile = File(...), file2: UploadFile = File(...)):
3      """
4      Compare two face images and determine if they match.
5
6      Args:
7          file1 (UploadFile): The first uploaded image.
8          file2 (UploadFile): The second uploaded image.
9
10     Returns:
11         dict: {"result": 1} if faces match, {"result": 0} otherwise.
12     """
13     if face_matcher is None:
14         raise HTTPException(status_code=400, detail="Model is not initialized. Call /initialize first.")
15
16     try:
17         # Read the images
18         img1 = await file1.read()
19         img2 = await file2.read()
20
21         # Perform face matching
22         result = face_matcher.match_face(img1, img2)
23         if result is None:
24             raise HTTPException(status_code=400, detail="No face detected in one or both images.")
25
26         return {"result": 1 if result else 0}
27
28     except Exception as e:
29         raise HTTPException(status_code=500, detail=f"Internal error: {str(e)}")
30

```

Client Implementation

Below is a detailed explanation of how the **utils.py** module is implemented and how it integrates into the client-side code (**client.py**) for evaluating the face recognition model.

1. The utils.py Module

FacePairDataLoader Class:

- **Purpose:**
This class is designed to load and pair face images from different datasets (CALFW, CPLFW, and LFW). It provides static methods to read files containing image pair information and generate lists of paired image paths along with their corresponding labels (1 for a matching pair, 0 for a non-matching pair).
- **load_cacp_pairs:**
 - **Input:**
A file path containing pair information and the directory where images are stored.
 - **Process:**
The method reads the file line by line, pairing two consecutive lines. It splits each line to extract the image filename and label. The full path for each image is constructed by combining the directory path with the filename.

- **Output:**
Returns a tuple containing a list of image path pairs and a list of corresponding labels. The label is determined by checking if the second part of the split line is not '0' (indicating a matching pair).
- **load_lfw_pairs:**
 - **Input:**
A CSV file with LFW pair information and the base directory for LFW images.
 - **Process:**
After skipping the header line, it processes each subsequent line.
 - For positive pairs, it extracts the person's name and two image numbers, constructs the file paths accordingly, and assigns a label of 1.
 - For negative pairs, it extracts two different person names and their corresponding image numbers to build file paths, then assigns a label of 0.
 - **Output:**
Similar to the previous method, it returns a tuple with a list of image path pairs and a list of labels.

```

1  class FacePairDataLoader:
2      """
3      A class for loading and pairing face images from different datasets.
4      """
5
6      @staticmethod
7      def load_cacp_pairs(file_path: str, img_dir: str) -> Tuple[List[Tuple[str, str]], List[int]]:
8          """
9          Load and pair images from the CALFW/CPLFW datasets.
10         """
11         pairs, labels = [], []
12
13         with open(file_path, 'r', encoding='utf-8') as file:
14             lines = file.readlines()
15
16             for i in range(0, len(lines), 2):
17                 parts1 = lines[i].strip().split()
18                 parts2 = lines[i + 1].strip().split()
19
20                 img1 = os.path.join(img_dir, parts1[0])
21                 img2 = os.path.join(img_dir, parts2[0])
22
23                 pairs.append((img1, img2))
24                 labels.append(1 if parts1[1] != '0' else 0)
25
26         return pairs, labels
27
28     @staticmethod
29     def load_lfw_pairs(pairs_file: str, lfw_dir: str = 'data/lfw/lfw-deepfunneled') -> Tuple[List[Tuple[str, str]], List[int]]:
30         """
31         Load and pair images from the LFW dataset.
32         """
33         pairs, labels = [], []
34
35         with open(pairs_file, 'r', encoding='utf-8') as file:
36             for line in file.readlines()[1:]: # Skip header line
37                 elements = line.strip().split(',')
38
39                 if not elements[-1]: # Positive pair
40                     person, img_num1, img_num2 = elements[:3]
41                     img1 = os.path.join(lfw_dir, person, f"{person}_{img_num1.zfill(4)}.jpg")
42                     img2 = os.path.join(lfw_dir, person, f"{person}_{img_num2.zfill(4)}.jpg")
43                     label = 1
44                 elif len(elements) == 4: # Negative pair
45                     person1, img_num1, person2, img_num2 = elements
46                     img1 = os.path.join(lfw_dir, person1, f"{person1}_{img_num1.zfill(4)}.jpg")
47                     img2 = os.path.join(lfw_dir, person2, f"{person2}_{img_num2.zfill(4)}.jpg")
48                     label = 0
49                 else:
50                     continue # Skip unexpected format lines
51
52                 pairs.append((img1, img2))
53                 labels.append(label)
54
55         return pairs, labels

```

calculate_evaluation_metrics Function:

- **Purpose:**
This function calculates evaluation metrics for the face recognition system based on true labels and predicted labels.
- **Metrics Calculated:**
 - **Accuracy:**
The ratio of correctly predicted pairs (both true positives and true negatives) to the total number of pairs.
 - **FMR (False Match Rate):**
The rate at which non-matching pairs are incorrectly predicted as matching.
 - **FNMR (False Non-Match Rate):**
The rate at which matching pairs are incorrectly predicted as non-matching.
- **Process:**
It iterates over both true and predicted labels, counts true positives, true negatives, false positives, and false negatives, and then computes the metrics accordingly.

```
1 def calculate_evaluation_metrics(true_labels: List[int], predicted_labels: List[int]) -> Tuple[float, float, float]:
2     """
3     Calculate evaluation metrics for face recognition performance.
4     """
5     tp = sum(1 for actual, pred in zip(true_labels, predicted_labels) if actual == 1 and pred == 1)
6     tn = sum(1 for actual, pred in zip(true_labels, predicted_labels) if actual == 0 and pred == 0)
7     fp = sum(1 for actual, pred in zip(true_labels, predicted_labels) if actual == 0 and pred == 1)
8     fn = sum(1 for actual, pred in zip(true_labels, predicted_labels) if actual == 1 and pred == 0)
9
10    total = len(true_labels)
11    accuracy = (tp + tn) / total if total else 0
12    fmr = fp / (fp + tn) if (fp + tn) else 0
13    fnmr = fn / (fn + tp) if (fn + tp) else 0
14
15    return accuracy, fmr, fnmr
```

2. The client.py Module

Purpose:

This module is responsible for interfacing with the FastAPI server that runs the face recognition model. It handles the initialization of the model on the server, sends image pairs for prediction, and evaluates the model using the provided dataset.

Key Functions:

- **initialize_model:**
 - **What It Does:**
Sends a POST request to the server's /initialize endpoint with the model name (e.g., "buffalo_1").

- **Outcome:**

If successful, the model on the server is initialized, and a confirmation message is printed. In case of an error, it raises a RuntimeError.

```
1 # URL of the FastAPI server
2 SERVER_URL = "http://127.0.0.1:8000"
3
4 def initialize_model(model_name: str) -> None:
5     """
6     Initialize the face recognition model on the FastAPI server.
7
8     Args:
9         model_name (str): The name of the model to initialize (e.g., 'buffalo_l').
10
11     Raises:
12         RuntimeError: If the server responds with an error.
13     """
14     url = f"{SERVER_URL}/initialize"
15     payload = {"model_name": model_name}
16     response = requests.post(url, params=payload)
17
18     if response.status_code == 200:
19         print(response.json().get("message", "Model initialized successfully.))
20     else:
21         error_msg = response.json().get("detail", "Unknown error")
22         raise RuntimeError(f"Error: {error_msg}")
23
```

- **predict:**

- **What It Does:**

Sends two image files to the server's /predict endpoint for comparison.

- **Process:**

Opens the image files in binary mode, then uses the requests library to send them as a multipart/form-data request.

- **Outcome:**

Receives a response indicating whether the images match (1) or not (0). If the server responds with an error, it raises a RuntimeError.

```
1 def predict(image_path1: str, image_path2: str) -> int:
2     """
3     Send two images to the FastAPI server for face recognition prediction.
4
5     Args:
6         image_path1 (str): Path to the first image.
7         image_path2 (str): Path to the second image.
8
9     Returns:
10         int: 1 if the server indicates a match; 0 otherwise.
11
12     Raises:
13         RuntimeError: If the server responds with an error.
14     """
15     url = f"{SERVER_URL}/predict"
16     with open(image_path1, "rb") as file1, open(image_path2, "rb") as file2:
17         files = {"file1": file1, "file2": file2}
18         response = requests.post(url, files=files)
19
20     if response.status_code == 200:
21         return int(response.json().get("result", 0))
22     error_msg = response.json().get("detail", "Unknown error")
23     raise RuntimeError(f"Error: {error_msg}")
```

- **main:**
 - **Workflow:**
 1. **Initialization:**
It starts by initializing the model on the server using the `initialize_model` function.
 2. **Data Loading:**
Based on the dataset specified (lfw, calfw, or cplfw), it uses the appropriate method from the `FacePairDataLoader` class in **utils.py** to load the image pairs and labels.
 3. **Prediction Loop:**
Iterates over each image pair, sends them to the server using the `predict` function, and collects the predicted labels. If an error occurs for a particular pair (for instance, if no face is detected), that pair is skipped.
 4. **Evaluation:**
After all predictions, it calculates the evaluation metrics (accuracy, FMR, FNMR) using the `calculate_evaluation_metrics` function and prints the results.

```

1  def main(model_name: str, dataset: str) -> None:
2      """
3      Main function to initialize the model, load image pairs from the chosen dataset,
4      perform predictions, and evaluate the performance of the face recognition model.
5
6      Args:
7          model_name (str): The name of the model to be initialized.
8          dataset (str): The dataset to use ('lfw', 'calfw', or 'cplfw').
9      """
10     try:
11         initialize_model(model_name)
12     except RuntimeError as e:
13         print(e)
14         return
15
16     data_loader = FacePairDataLoader()
17
18     dataset_paths = {
19         "lfw": ("data/lfw/pairs.csv", data_loader.load_lfw_pairs),
20         "calfw": ("data/calfw/pairs_CALFW.txt", data_loader.load_cacp_pairs),
21         "cplfw": ("data/cplfw/pairs_CPLFW.txt", data_loader.load_cacp_pairs),
22     }
23
24     if dataset.lower() not in dataset_paths:
25         print("Unsupported dataset. Please choose from 'lfw', 'calfw', or 'cplfw'.")
26         return
27
28     file_path, loader_func = dataset_paths[dataset.lower()]
29     img_dir = f"data/{dataset.lower()}/aligned images" if dataset.lower() != "lfw" else None
30     pairs, labels = loader_func(file_path, img_dir=img_dir) if img_dir else loader_func(file_path)
31
32     predicted_labels: List[int] = []
33     for img1, img2 in pairs:
34         try:
35             predicted_labels.append(predict(img1, img2))
36         except RuntimeError as e:
37             print(f"Skipping pair ({img1}, {img2}) due to error: {e}")
38
39     accuracy, fmr, fnmr = calculate_evaluation_metrics(labels, predicted_labels)
40     print(f"Model Evaluation - Accuracy: {accuracy:.4f}, FMR: {fmr:.4f}, FNMR: {fnmr:.4f}")
41

```

Command Line Interface:

The module also uses the argparse library to allow users to specify the model and dataset via command-line arguments when running the script.

```
1 if __name__ == "__main__":
2     parser = argparse.ArgumentParser(description="Face Recognition Model Evaluation")
3     parser.add_argument("--model", type=str, required=True, help="Face recognition model (e.g., 'buffalo_1')")
4     parser.add_argument("--dataset", type=str, required=True, help="Dataset to use: 'lfw', 'calfw', or 'cplfw'")
5     args = parser.parse_args()
6     main(args.model, args.dataset)
```

Results

In this section, we present the quantitative performance of our face recognition models, **buffalo_s** and **buffalo_l**, on three datasets: LFW, CALFW, and CPLFW. We report key metrics including Accuracy, False Match Rate (FMR), and False Non-Match Rate (FNMR) to evaluate each model's effectiveness. The table below summarizes these metrics for each model and dataset, along with remarks on their performance.

Model	Dataset	Accuracy (%)	FMR (%)	FNMR (%)
buffalo_s	LFW	98.23	0.27	2.43
buffalo_s	CALFW	94.07	2.2	9.67
buffalo_s	CPLFW	90.4	0.7	20
buffalo_l	LFW	98.65	0.63	2.8
buffalo_l	CALFW	95.3	1.01	9
buffalo_l	CPLFW	94.81	1.8	10.01

Explanations:

- **Accuracy (%):**
Represents the overall percentage of correctly classified face pairs. A higher value indicates better recognition performance.
- **False Match Rate (FMR %):**
Measures the percentage of non-matching pairs that are incorrectly recognized as matches. Lower values are preferable, as they indicate fewer false positives.
- **False Non-Match Rate (FNMR %):**
Indicates the percentage of matching pairs that are incorrectly classified as non-matches. A lower FNMR signifies better sensitivity in recognizing genuine matches.