VPBK - Projekt

None

Dominik Lichnovský

None

Table of contents

1. VPBK Project

1.1 Zadání

- Úkolem je implementovat projekt deepface a napsat dokumentaci.
- Bude se kontrolovat funkčnost a způsob ověření funkčnosti
- Podle možností programu budou testovány buď natrénované/registrované tváře/osoby vůči novým obrázkům + vaše vlastní tvář (registrovat do systému a následně otestovat).

1.2 Rozvržení projektu

```
main.py                                 # Main Python File
requirements.txt                        # Python App Requirements
requirements-dev.txt                    # Python Developer Requirements
ui/
    resources/                          # Folder with resources
            deepface-icon-labeled.png   # DeepFace Project Icon
    main_window.py                      # UI Main Window Python File
    main_window.ui                      # UI Main Window Qt Designer File
mkdocs.yml                              # The configuration file.
docs/
    index.md                            # Homepage
    deepface.md                         # DeepFace Documentation
    implementace.md                     # Implementation Documentation
    code/
        deepfacehelper.md               # Documentation for the DeepFaceHelper class.
        gui.md                          # Documentation for the GUI class.
```

1.3 Zdroje

- DeepFace

2. DeepFace

Deepface je framework pro rozpoznávání tváří a analýzu obličejových vlastností (věk, pohlaví, emocí a rasy) pro jazyk Python.

Obsahuje modely:

- VGG-Face
- FaceNet
- OpenFace
- DeepFace
- DeepID
- ArcFace
- Dlib
- SFace
- GhostFaceNet.

2.1 Instalace

Pro instalaci je potřeba mít nainstalovaný Python 3.8 nebo novější. Doporučuji pracovat v systému Linux vzhledem k limitaci a verze tensorflow package, ale je možné použít i Windows.

Framework se vyskytuje pod názvem `deepface` na PyPI.

2.2 Vlastnosti/Funkce

- Rozpoznávání tváří

Tato funkce umožňuje rozpoznat tváře na 2 obrázcích a verifikovat tak, pokud se jedná se stejné nebo jiné osoby.

- Analýza tváří

Tato funkce aplikuje několikrát rozpoznání tváří. Porovná obrázek se svojí databází a na základě toho vrátí list s výsledky.

- Embeddings

Tato funkce umožňuje získat vektorové reprezentace tváří z obrázků.

- Analýza atributů tváře

Tato funkce umožňuje získat informace o věku, pohlaví, rasě a emocích z obrázků tváří.

- Backendy detektoru tváří

DeepFace implementuje mnoho backendů pro detekci tváří.

```
* Můžete si vybrat mezi:
    * OpenCV
    * Dlib
    * Ssd
    * MTCNN
    * fastmtcnn
    * RetinaFace
    * mediapipe
    * yolov8
    * yunet
    * centerface
```

- Analýza v reálném čase

Knihovna má v sobě zabudované funkce pomocí kterých můžete analyzovat tváře v reálném čase např. přes webkameru.

- API

Aby nemuselo všechno běžet na jednom stroji, je možné použít API, které běží na serveru a poskytuje výsledky zpracování obrázků vzdáleně. Je zde dostupný i Docker image, takže je možné provozovat několik instancí ve velkém clusteru např. přes Kubernetes.

- CLI

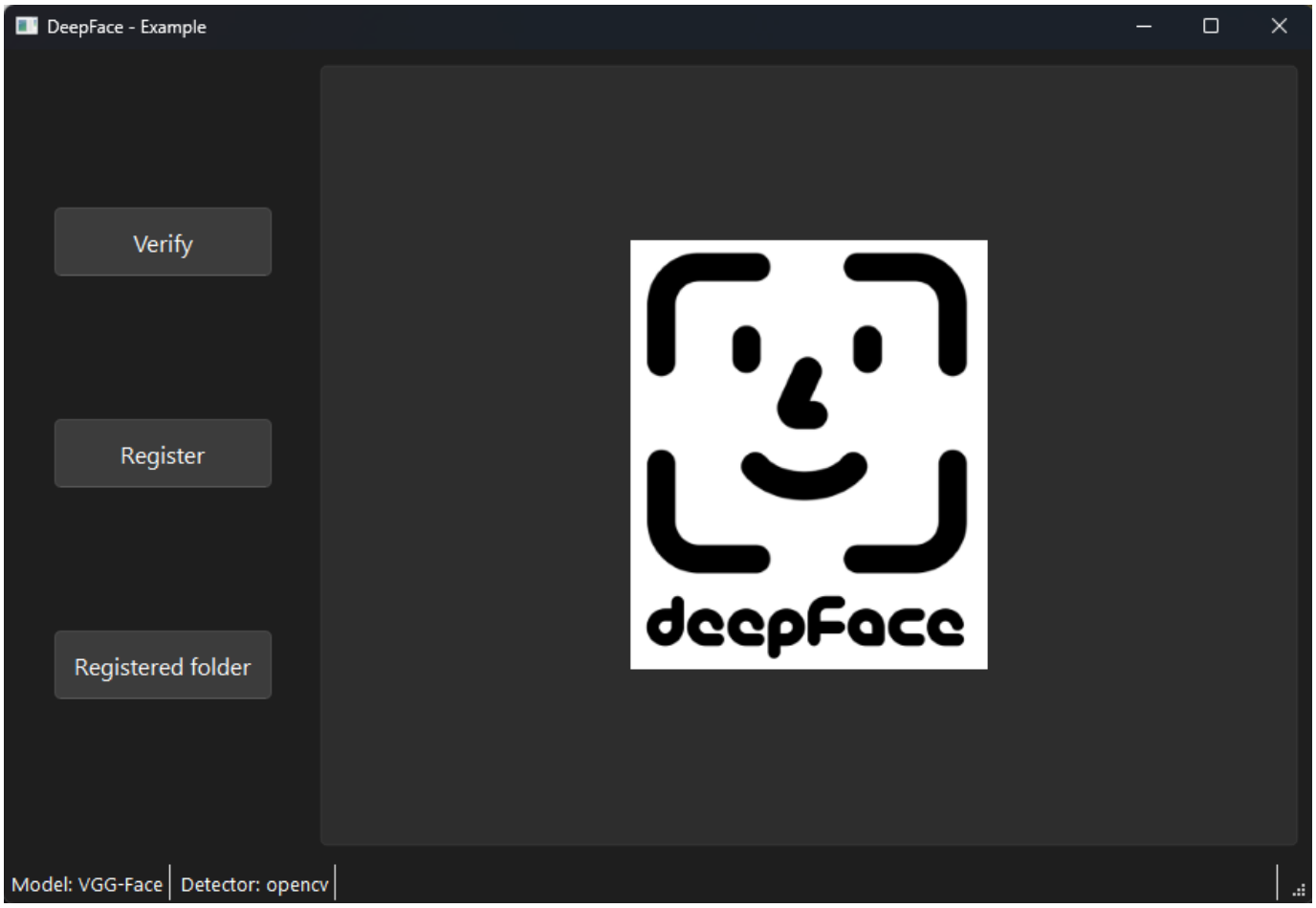
Knihovna obsahuje i CLI, které umožňuje spouštět různé funkce z příkazové řádky.

3. Implementace

Pro demonstraci využití knihovny jsem vytvořil jednoduchý program, který umožňuje rozpoznat tváře a zaregistrovat tváře do databáze přes webkameru.

Aplikace neklade důraz na UI/UX, ale spíše na demo funkcionalitu knihovny.

Screenshot aplikace:



3.1 Ovládání

Aplikace obsahuje 3 tlačítka:

- **Verify**
- Spustí webkameru a umožní ověřit tváře na základě databáze tváří.
- Backend začne analyzovat každý snímek z kamery a snaží se na něm najít tváře. Zvýraznění tváře obdelníkem znamená, že na snímku byla rozeznána tvář. Pokud nalezne aspoň 30 tváří (nastavený defaultní threshold), tak se pokusí pomocí vybraného modelu porovnat tváře s databází tváří.
- Pokud nalezne shodu, tak se snímání vypne a zobrazí se výsledek (Identita, Distance, Threshold) a v posledním snímku se zvýrazní nalezená tvář s identitou.
- Pokud se shoda nenalezne, tak se zobrazí ve StatusBaru "No identity found." a snímání znovu pokračuje s nulovým thresholdem pro nalezání tváří.
- Verifikaci můžeme manuálně zastavit pomocí stejného tlačítka.
- **Register**
- Spustí webkameru a umožní zaregistrovat tvář do databáze tváří.
- Po spuštění webkamery lze pomocí tlačítka "Take picture" vyfotit tvář. Po vyfocení se zobrazí výsledek a je povolí se tlačítko "Register picture". Po kliknutí na tlačítko uživatel může vybrat pod jakou identitou má být tvář zaregistrována, případně vytvořit novou identitu.
- Pokud se snímek nevydaří, tak je možné ho vyfotit znovu pomocí tlačítka "Retake picture".
- **Registered folder** - Otevře složku databáze s registrovanými tvářemi.

4. Code DeepFaceHelper

DeepFace - Example

4.1 `DeepFaceHelper`

A helper class for handling facial recognition tasks using the DeepFace library.

Parameters:

Name	Type	Description	Default
`db_path`	`str`	The path to the database of faces to be recognized.	*required*
`model_name`	`str`	The name of the model to be used for face recognition.	`'VGG-Face'`
`detector_backend`	`str`	The backend to be used for face detection.	`'opencv'`
`distance_metric`	`str`	The metric to be used for measuring similarity.	`'cosine'`
`source`	`int`	The source of the video capture.	`0`
`frame_threshold`	`int`	The threshold for the number of frames with faces.	`30`

4.1.1 `capture_camera()`

Capture a frame from the camera and display it.

Returns:

Type	Description
	None

4.1.2 `capture_camera_and_analyze_face(face_found)`

Capture camera and analyze face.

This method captures the frame from the camera and performs face analysis on it. If a face is found, it performs facial recognition and updates the status bar with the identity result.

Parameters:

Name	Type	Description	Default
`face_found`	`bool`	A flag indicating whether a face has been found in the previous frame.	*required*

Returns:

Type	Description
`None`	None

4.1.3 `face_analysis()`

This method is responsible for building the facial recognition model and initializing the embeddings.

Note: The detected face is a 224x224x3 numpy array of zeros, which serves as a placeholder for the actual face data.

4.1.4 `load_registered()`

Load the names of the registered faces from the database.

This method lists the names of the directories in the database path, each of which represents a registered face.

Returns:

Name	Type	Description
`list`		A list of the names of the registered faces.

4.1.5 `perform_facial_recognition_details(img, faces_coordinates, detected_faces, db_path=None, detector_backend=None, distance_metric=None, model_name=None)`

Perform facial recognition on the detected faces.

This method takes an image, the coordinates of the faces in the image, and the detected faces themselves, and performs facial recognition on each face. It uses the specified database path, detector backend, distance metric, and model name, or defaults to the ones specified in the class instance if not provided.

Parameters:

Name	Type	Description	Default
`img`	`ndarray`	The image to perform facial recognition on.	*required*
`faces_coordinates`	`list`	A list of tuples containing the coordinates of each face in the image.	*required*
`detected_faces`	`list`	A list of the detected faces as numpy arrays.	*required*
`db_path`	`str`	The path to the database of faces to be recognized. Defaults to the one specified in the class instance.	`None`
`detector_backend`	`str`	The backend to be used for face detection. Defaults to the one specified in the class instance.	`None`
`distance_metric`	`str`	The metric to be used for measuring similarity. Defaults to the one specified in the class instance.	`None`
`model_name`	`str`	The name of the model to be used for face recognition. Defaults to the one specified in the class instance.	`None`

Returns:

Type	Description
`IdentityResult | None`	IdentityResult | None: An instance of the IdentityResult dataclass containing the identified image path, the identified image, the distance, and the threshold, or None if no identity was found.

4.1.6 `register_face(folder_name)`

Register a face in the database.

This method takes a folder name as input and checks if a directory with that name exists in the database. If it doesn't, it creates one. Then, it encodes the currently captured image (stored in `self.register_img`) into PNG format and calculates its SHA-256 hash. Finally, it saves the image in the newly created directory, using the hash as the file name.

Parameters:

Name	Type	Description	Default
`folder_name`	`str`	The name of the folder to create in the database.	*required*

Returns:

Type	Description
	None

4.1.7 `search_identity_details(detected_face, db_path, model_name, detector_backend, distance_metric)` `staticmethod`

Search an identity in facial database.

Parameters:

Name	Type	Description	Default
`detected_face`	`ndarray`	extracted individual facial image	*required*
`db_path`	`string`	Path to the folder containing image files. All detected faces in the database will be considered in the decision-making process.	*required*
`model_name`	`str`	Model for face recognition. Options: VGG-Face, Facenet, Facenet512, OpenFace, DeepFace, DeepID, Dlib, ArcFace, SFace and GhostFaceNet (default is VGG-Face).	*required*
`detector_backend`	`string`	face detector backend. Options: 'opencv', 'retinaface', 'mtcnn', 'ssd', 'dlib', 'mediapipe', 'yolov8', 'centerface' or 'skip' (default is opencv).	*required*
`distance_metric`	`string`	Metric for measuring similarity. Options: 'cosine', 'euclidean', 'euclidean_l2' (default is cosine).	*required*

Returns:

Name	Type	Description
`identity_result`	`IdentityResult`	identified image path, identified image, distance and threshold

4.2 `IdentityResult` `dataclass`

A dataclass representing the result of an identity search.

Attributes:

Name	Type	Description
`identified_image_path`	`str`	The path to the identified image.
`identified_image`	`ndarray | None`	The identified image as a numpy array, or None if no image was found.
`distance`	`float`	The distance between the identified face and the face in the database.
`threshold`	`float`	The threshold for considering a match.

5. Code GUI

DeepFace - Example

5.1 `MainWindow`

Bases: `QMainWindow`

The MainWindow class inherits from QMainWindow and represents the main window of the application.

This class is responsible for setting up the user interface, handling user interactions, and managing the application's state. It contains methods for handling button click events, displaying images, and managing the state of the application.

5.1.1 `register_picture_pushbutton_click()`

Handle the click event of the register picture button.

This method loads the names of the registered faces from the database and displays a dialog for the user to choose or input an option. If the user confirms their choice or input, the method registers the face under the chosen or inputted name.

Returns:

Type	Description
	None

5.1.2 `register_pushbutton_click()`

Handle the click event of the register button.

Returns:

Type	Description
	None

5.1.3 `registered_pushbutton_click()`

Handle the click event of the registered button.

This method opens the directory containing the database of registered faces using the default file explorer of the operating system.

Returns:

Type	Description
	None

5.1.4 `reset_camera_label()`

Reset the camera label to its default state.

Returns:

Type	Description
	None

5.1.5 `reset_register_button()`

Reset the register button to its default state.

Returns:

Type	Description
	None

5.1.6 `reset_register_frame()`

Hide the register frame in the GUI.

Returns:

Type	Description
	None

5.1.7 `reset_verify_button()`

Reset the verify button to its default state.

Returns:

Type	Description
	None

5.1.8 `show_opencv_img(img)`

Display an OpenCV image in the GUI.

Parameters:

Name	Type	Description	Default
`img`	`ndarray`	The OpenCV image to be displayed.	*required*

Returns:

Type	Description
	None

5.1.9 `start_verify_manually()`

Start the manual verification process.

Returns:

Type	Description
	None

5.1.10 `stop_verify_identified()`

Stop the verification process after an identity has been found.

Returns:

Type	Description
	None

5.1.11 `stop_verify_manually()`

Stop the manual verification process.

Returns:

Type	Description
	None

5.1.12 `take_picture_pushbutton_click()`

Handle the click event of the take picture button.

Returns:

Type	Description
	None

5.1.13 `verify_pushbutton_click()`

Handle the click event of the verify button.

Returns:

Type	Description
	None

5.2 `ButtonState`

Bases: `Enum`

An enumeration representing the state of a button.

Attributes:

Name	Type	Description
`INACTIVE`	`int`	The state representing a button that is not currently being interacted with.
`ACTIVE`	`int`	The state representing a button that is currently being interacted with.

5.3 Sources

- Qt Docs