SCALABLE PERSON IDENTIFICATION SYSTEM FOR REAL-TIME APPLICATIONS

Martin Rajnoha
Doctoral Degree Programme (3), FEEC BUT
E-mail: martin.rajnoha@vutbr.cz

Supervised by: Radim Burget
E-mail: burgetrm@feec.vutbr.cz

Abstract: Face recognition systems can play significant role in our every day lives. This paper proposes a scalable system for person identification based on face recognition methods and its implementation that utilizes queues, containers and microservices architecture. The proposed system uses a GPU acceleration therefore it can run in real-time. It utilizes two deep neural networks - Single Shot Multibox Detector (SSD) for a face detection and Facenet for a face recognition.

Keywords: facerecognition, scalable, realtime, detection, microservices, queues, identification

1 INTRODUCTION

In the last decades the security situation all around the world has become more unpredictable and full of threats in the meaning of terrorism, robberies etc., especially at crowded places. Because of that pedestrian identification system can play very significant role. Thanks to latest technologies and improvements in information technologies, mainly fields such as computer vision, artificial intelligence and deep learning, it is possible to build systems that can minimize mentioned dangers and may significantly support security services. The amount of people walking through the shopping malls or airports is enormous and real-time control of all these people with list of suspects is almost impossible task for human [1].

On the other hand, this task can be relatively easily solved by computer programs. This paper describes a whole system and its implementation for real-time person identification purposes based on face recognition. The proposed person identification system can be deployed on already existing camera systems and it can utilize and extend their abilities. The system utilizes deep learning technology, especially convolutional neural networks [2] for both face detection and face recognition parts. Because of that it is supposed to be more precise, robust and resistant to camouflage (glasses, hat…). It does not store any sensitive and personal information about the people. Identification is performed in real-time and it uses embeddings [3] that are created from face by one-way function without chance of reconstruction of an original image. A database of suspects contains also only embeddings. The system does not have to be used only for suspects identification but it can be used also as a kind of authorization system in protected areas and notify security service when there is an unknown person detected.

The rest of the paper is structed as follows: Section 2 deals with a methodology, presents a whole system and briefly describes applied methods. This part is focused on a structure and implementation of the system (microservices, queues, scalability). After that there is a result Section 3 that presents outcomes regarding the proposed system and the last Section 4 concludes the paper.

2 METHODOLOGY

The system described in this paper is supposed to run in real-time, therefore delay requirements are critical and it must be robust enough regarding its wide range of possible applications. Conventional face recognition methods are basically designed for handling ideal cases such as one face on the image in a good quality. Of course, there is some robustness and ability to handle worse cases but they are not primarily designed for multiple face handling. Because of that the overall system can be divided into two main components that are face detection and face recognition. Both are stand-alone independent components but in the scope of the overall system they have to be performed sequentially because of the face recognition requires the face detection.

The first part of process is detection of faces using Single Shot Multibox Detector (SSD) [4] method that utilizes deep convolutional neural network. It detects all faces on the image using only one "shot" instead of conventional detectors that must process the same image multiple times (different scales). A transfer learning [5] method was used and an existing model (its weights) was fine-tuned to perform the face detection. Each particular face is then cropped based on detections of SSD and embedding vector of length 128 is created from the face. Another deep neural network – Facenet [3] is used for face encoding to the embedding vector, i.e. finding the most important features of the face. After that the resulting embedding is compared (using Euclidean distance) with embeddings in the database.

Any face recognition system struggles with different angle of persons look (angle of the camera). A face landmark estimation [6] method is used for partial elimination of this problem. Before embeddings creation, the face is centered as much as possible using 68 points that contains each face and image processing trasformations (rotations, shear . . .) [7]. The overall system is obviously more complicated (see Fig. 1). Components have to be connected, the system contains the database, it must have some error handling and of course it must have some application interface that connects the system with an user interface.

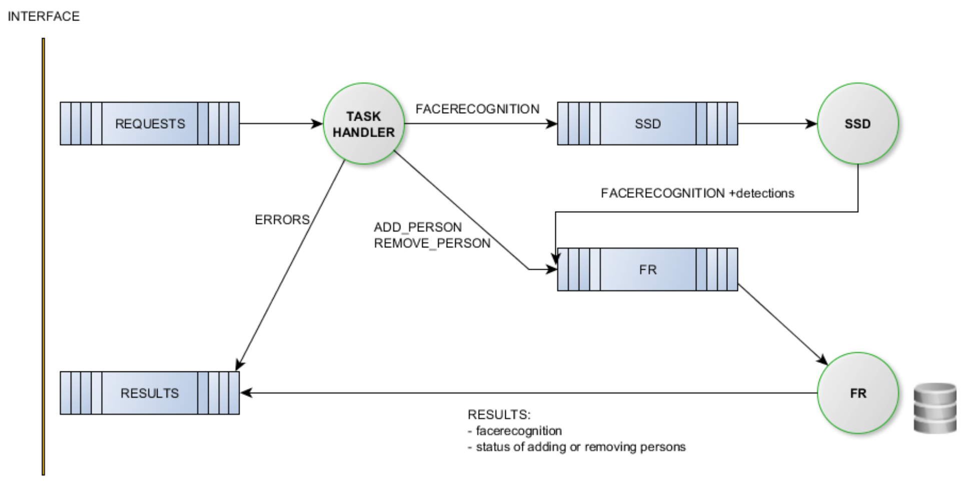

Figure 1: An architecture of the overall system for person identification.

The system is designed to handle three tasks: *facerecognition*, *add person*, *remove person* and it returns an answer in JSON format (coordinates of faces and percentual match with suspects). Next subsections briefly describe implementation of mentioned components and a communication between them. They also deal with used tools and techniques that all together creates a robust and scalable person identification system.

2.1 MICROSERVICES

The proposed system uses a microservices architecture[1] that divides an application into multiple services that are independently deployable, maintainable and testable. Previously mentioned components – the face detection (SSD) and the face recognition (FR) are considered as individual services and there is one more service called "task handler" which is responsible for error handling and routing of tasks. Each service runs as a container using Docker[2] platform. Docker does not allocate any hardware resources but uses them directly during runtime. Each docker image has its own environment and all necessary dependencies must be installed and source code and files have to be copied in it. Once the image is built, it can be run instantaneously (possible in more instances).

The task handler service is responsible for tasks reading and controlling of their correct behaviour. The task must contain the image in BASE64 format and type of the task definition. If the task is correct, task handler sends it to the corresponding service, otherwise the task is returned with some error description. The SSD service gets the input image in BASE64 and returns JSON with coordinates of face detections. The SSD uses pre-trained machine learning model that is part of the container and it is loaded immediately after start of the container. The face recognition service can handle two types of tasks: The first one is facerecognition that takes coordinates of face detections as the input and returns percentual match with suspects for each detection. The second type of task is a management task in the meaning of adding or removing persons from the database. Because of the requirement of fast processing, the whole database is loaded into the RAM memory after the container starts. This database is mounted out of the container to the hard-disk in the case of failure. Any change in the database is then synchronized between the RAM memory and the hard-disk copy.

2.2 QUEUES

Individual services in form of containers were described in the previous subsection. But how do they communicate between each other? A queue system is used for this reason and specifically RabbitMQ[3] for the case described in this paper. Using queues in general has many advantages such as: It can ensure communication between different services. It creates an interface between services and it does not matter what programming language or what operating system is used by services. It can have multiple producers and multiple consumers. Tasks are dynamically added and fetched, basically in FIFO (first in first out) order. A queue size can be monitored and based on utilization there can be a program for limitation of input tasks amount. Queues does not provide communication only between services (containers) but they can provide an application interface to frontend or user interface. It is simpler for debugging and updating or exchange of some service can be done without an outage.

2.3 SCALABILITY

The biggest advantage of the proposed architecture is its scalability. As it was already mentioned queues can have multiple consumers and each service in the container can run in multiple independent instances. It allows to exchange any service in any time. First, a new service is connected to the queue and it starts to consume tasks. Both services can run simultaneously and there is also time for testing. if everything works properly, the old service is stopped (container killed) and after that only new service consumes and handles tasks.

In the case the system is not able to run in real-time, there is a possibility to start more containers (workers). An example of running multiple workers is in Fig. 2 where there are two instances of SSD and three instances of Facerecognition service.

[1]`https://microservices.io/`
[2]`https://www.docker.com/`
[3]`https://www.rabbitmq.com/`

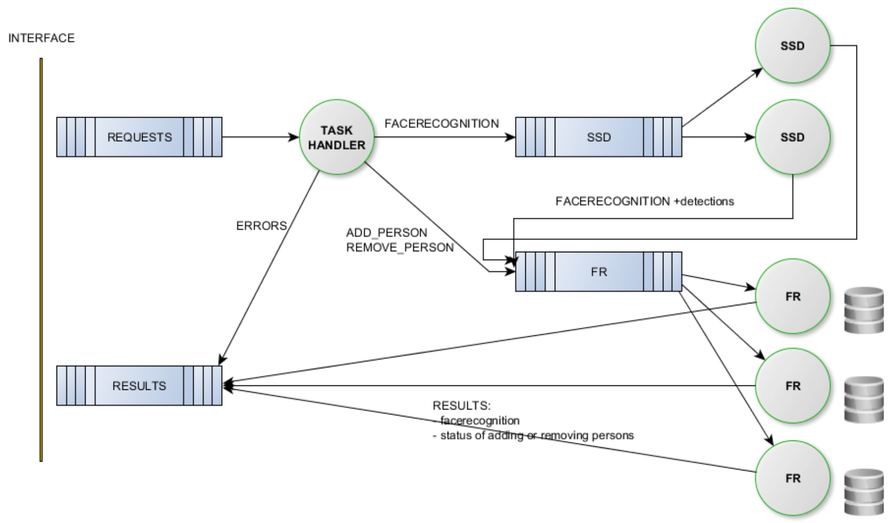

Figure 2: The usage of multiple instances of container services.

Running multiple Facerecognition containers is problematic because of a synchronization of the database between all containers. All containers are mounted to the same database on the hard-disk but they run their own instances of the database in the RAM memory. If there is a task for the database management (add or remove person), only one container takes this task and processes it (it updates the RAM and the database on the hard-disk). Other containers do not know anything about this change in the database. The Facerecognition service has its own management queue for handling this problem. After the container finishes database update it sends copy of the task to all other container queues with a flag that says do not update the database on the hard-disk (it would cause an infinite loop).

3 RESULTS

The critical part is an ability to run in real-time. There is a possibility to use multiple threads for parallelization of services. This makes sense only when a CPU is used and given service can be faster as many times as number of CPU cores. Because of the proposed system uses two deep neural networks, a GPU acceleration is more significant instead of the CPU parallelization (see Table 1). A conventional face recognition framework (e.g. face-recognition[7]) uses Histogram of Oriented Gradients (HOG) [8] method for the face detection which runs only on the CPU. A comparison of average times using the CPU and the GPU for face detections and embeddings creation by this framework is in Table 1.

Operation	**CPU [s]**	**GPU [s]**
SSD detection	0.76	0.02
Face embeddings	0.35	0.01
HOG detection and face embeddings	1.26	0.35
SSD detection and face embeddings	-	0.031

Table 1: Comparison of average times on CPU and GPU (Titan Xp) for different operations

The last record of the Table 1 represents an average time for the face detection by the SSD and consequently embeddings creation. The average time 0,031 sec presents how powerful and effective are deep neural networks with the GPU acceleration. As it was already mentioned in Section 2.3, multiple workers (instances of services) can be used to process tasks in parallel manner instead of a parallelization within individual process. Processing time of the SSD is 40 fps for 1 worker and 60 fps for 2 workers. It is caused by high hardware requirements and whole SSD network is not fitted twice to the GPU memory. A face recognition processing time is 13 fps for 1 worker, so 4 workers are needed for 50 fps.

4 CONCLUSION

This paper deals with challenging system of a real-time person identification based on face recognition methods. It describes its two most important components - Single Shot Multibox Detector (SSD) for the face detection and Facenet for the face recognition. The paper is focused on the implementation of the proposed system mainly the ability to run in real-time. It describes microservices architecture built on Docker platform and communication using a queue system (RabbitMQ). It presents advantages of a proposed structure mainly in the matter of deployment, testing and scalability. Several test cases and measurements were performed in scope of this work that are summarized in the Section 3. Average processing time of the proposed system is 0.031 sec using only one instance of each service. Processing speed of 50 fps can be achieved by running two SSD service instances (containers) and four instances of the Facerecognition service. No personal data is stored stored during processing and the database of suspects contains only embeddings created by one-way function from faces. The proposed system runs in real-time and thanks its architecture can be easily applied even for older and existing camera systems.

ACKNOWLEDGEMENT

Research described in this paper was financed by NSP LO1401 and by Ministry of the Interior of the Czech Republic grant VI20172019086. For the research, infrastructure of the SIX Center was used.

REFERENCES

[1] Rajnoha M., Povoda L., Masek J., Burget R., Dutta M.K.: Pedestrian Detection from Low Resolution Public Cameras in the Wild. In: 5th International Conference on Signal Processing and Integrated Networks (SPIN) 2018, pp. 291-295. IEEE.

[2] Krizhevsky, A., Sutskever, I., Hinton, G. E: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. 2012, pp. 1097-1105.

[3] Schroff F., et al.: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 815–823.

[4] Liu W., et al.: SSD: Single Shot MultiBox Detector. In: Computer Vision – ECCV 2016, Springer International Publishing, 2016, pp. 21–37.

[5] Pan, S. J., Yang, Q.: A survey on transfer learning. In: IEEE Transactions on knowledge and data engineering 22, no. 10, 2010, pp. 1345-1359.

[6] Kazemi V., Sullivan, J.: One millisecond face alignment with an ensemble of regression trees. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 1867–1874.

[7] Geitgey A.: Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning. Medium.com, 2016

[8] Dalal N., Triggs B.: Histograms of Oriented Gradients for Human Detection. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2015.