COMP4651 Project: Serverless Image Face-Blurring

By: Young, James Yang

SID: 20740589

1. Introduction

In an era characterized by the widespread use of digital media, privacy alongside the protection of personal information has become an increasing issue. One way of safeguarding privacy is through blurring faces in images, which ensures that individuals' identities remain anonymous. However, for individuals who wish to post images to the internet but have limited computer or technical knowledge, blurring faces in images can be difficult. Although there are existing software and websites that can automate face blurring, they often use closed-source tools [1]. Therefore, this project aims to build a serverless image face blurring application using open-source alternatives.

2. System Overview

This project follows closely to a serverless image recognition and processing backend. The main tools used to replace AWS applications are OpenFaaS [2] in place of Lambda for serverless functions, MinIO [3] in place of S3 for storage bucket, and MTCNN [4] in place of Rekognition for face detection. This was deployed on Minikube [5] to run on a single local machine. For demonstration purposes, a front-end was also written in Flask [6] but was not deployed as part of the Kubernetes cluster.

2.1 Architecture

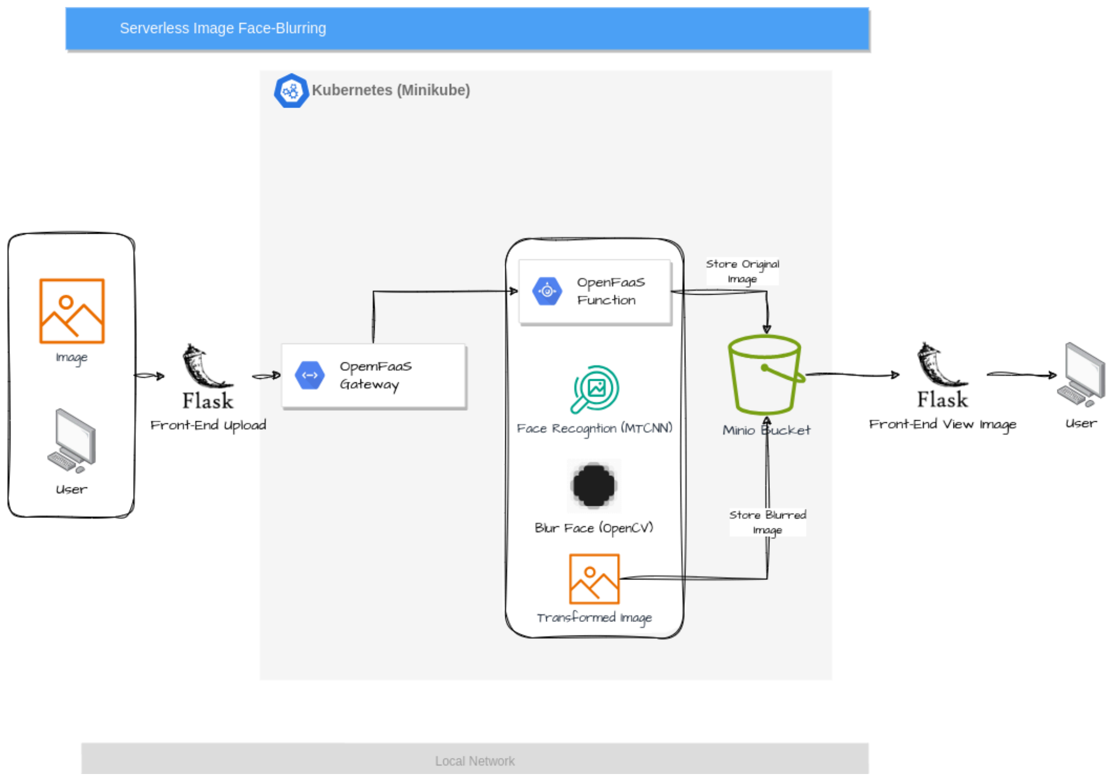

Figure 1: Serverless Application Diagram

Figure 1 shows the overall architecture. The application is set up with Kubernetes (Minikube) for orchestration. I use the OpenFaaS platform as it packages all the Python packages needed including MTCNN for face detection, OpenCV for blurring the section of the image containing the face, and MinIO's Python SDK for storing the original and blurred image into the bucket. It is packaged into a Docker container and can be triggered through a gateway. The function requires that the image be converted to base64 before sending the encoded base64 to the OpenFaaS gateway as I encountered errors reading raw binary data and was unable to resolve it. After receiving the encoded base64, I decode it, use MTCNN for face detections to obtain the regions to blur, then apply Gaussian blur using OpenCV.

Then, after blurring any detected faces on the image, I save both the original and transformed image in separate folders in a S3 bucket. As mentioned previously, I used MinIO for the object storage system. As the storage is persistent, it means that the user can view and download the image from the bucket through the MinIO API.

For demonstration purposes, a front-end website is written with Flask to facilitate uploading images as base64 and also viewing images from the bucket once it has been uploaded and blurred. Whilst ideally it can also be deployed as part of the Kubernetes cluster, due to time constraints I was unable to complete this part.

3. Results and Analysis

Figure 2 shows a side-by-side comparison of the original and blurred image for both a single person and a small group. As the images show, the function is able to handle multi-person images quite well when the image quality of the faces are clear.

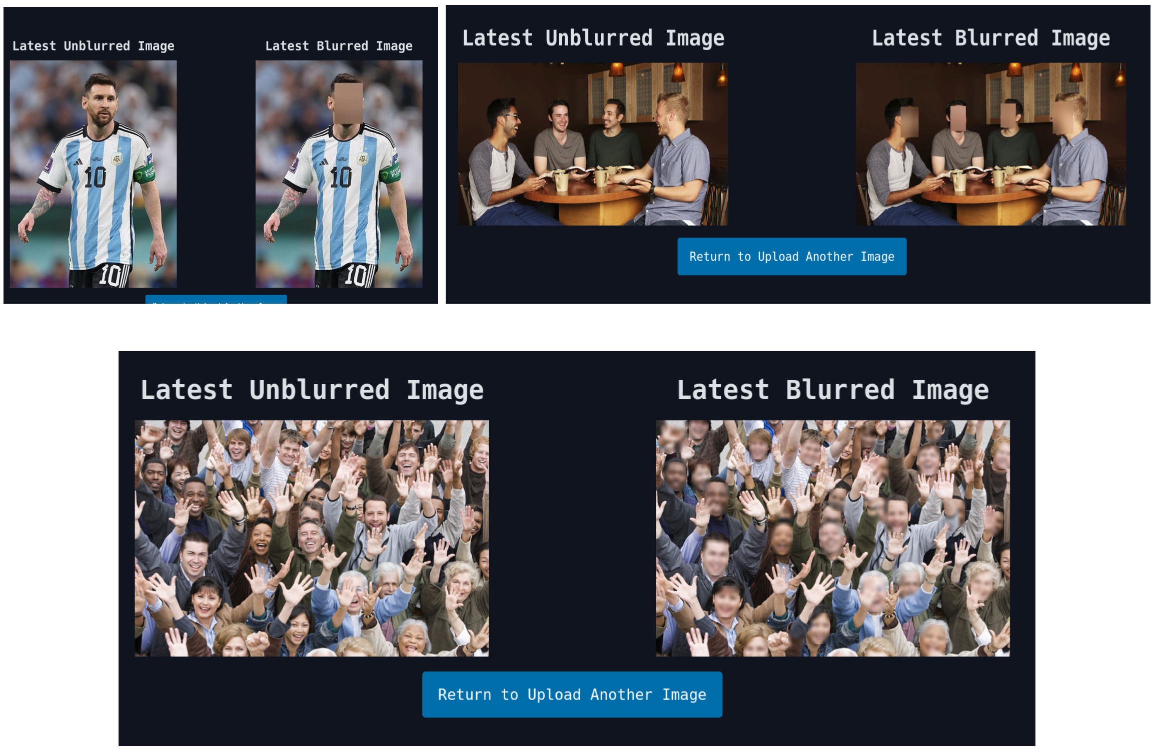

Figure 2: Original and blurred images from MinIO bucket

However, one of the issues with the current model is finding the balance between speed and accuracy. The current parameters have relatively good performance on images with less than 10 clear faces (such as the top 2 images in Figure 2), where it usually takes less than 5-10 seconds to finish the blurring. However, when using an image with a large crowd of people such as the bottom image in Figure 2, the function hangs for a while, usually taking over 20 seconds to complete the image detection and transformation depending on the number of people. Although lowering parameters such as scale factor could boost speed, I found it to lower the accuracy by a lot in large groups or in situations where the face is facing the side of the camera. Nevertheless, I believe the application still has acceptable performance for real-world use-cases.

4. Conclusion

The project was successful in building a serverless application that detects faces in an image, blurs them, and stores them in a storage bucket. It uses open-source alternatives to AWS applications such as OpenFaaS and is deployable to a Kubernetes cluster.

4.1 Closing Thoughts

Since I had never used most of the tools listed such as Kubernetes, MinIO, and so on, there were many technical challenges faced. For example, I spent a long time trying to connect to the MinIO API through the OpenFaaS function. I kept getting 502 Bad Gateway errors until later on I realized that using localhost:9000 inside the container means that it uses the container's IP, not my machine's IP. These types of issues were prevalent throughout my project, making it difficult to add more features that I wanted to add as I will mention in the next section.

4.2 Improvements and Future Goals

Due to my limited experience and time constraints, there are still many features that can be added to this project. The main feature I would have liked to add is video support. Another feature I would have liked to add is user authentication, which may include using a relational database for storing user information.

Furthermore, as mentioned previously, more fine-tuning and empirical testing can be done for better performance. I used MTCNN as it performed better than using OpenCV’s Cascade Classifier for face detection. With more time, I could try other models as well to compare their performance and speed. Lastly, I would like to package and deploy my front-end Flask application in Kubernetes for easier deployment.

References

[1] T. Das, “Face blurring software - 9 best options for your privacy,” MSPoweruser, https://mspoweruser.com/face-blurring-software/ (accessed May 16, 2024).

[2] “ Serverless Functions, Made Simple,” OpenFaaS, https://www.openfaas.com/ (accessed May 16, 2024).

[3] “ The Object Store for AI Data Infrastructure ,” MinIO, https://min.io/ (accessed May 16, 2024).

[4] D. Sandberg, “MTCNN,” GitHub, https://github.com/ipazc/mtcnn (accessed May 16, 2024).

[5] Minikube, https://minikube.sigs.k8s.io/ (accessed May 16, 2024).

[6] “Welcome to flask,” Flask, https://flask.palletsprojects.com/en/3.0.x/ (accessed May 16, 2024).