Software requirements specification for project CVIP

0. Authors

Kotenkov Maxim

Yakovleva Valeria

Makanin Kirill

1. Introduction

CVIP - Computer Vision Image Processing tool. CVIP will let users easily and efficiently edit photos and videos.

Using CVIP, users can resize, crop images, detect and blur faces on it, and perform other operations connected to CV with images and videos available with CVIP command line tool.

2. Glossary

Scale - the resizing of a digital image.

CV - Computer Vision.

Device’s firmware - the main system on the device.

Multiprocessing - the ability of a program to use the pair or more physical and virtual processors in one system.

GUI - Graphical user interface, that is used for communication between user and the tool.

GitHub - is a platform and cloud-based service for software development and version control, allowing developers to store and manage their code.

Repository - a collection of files and history on Github.

Pull Request - a way to suggest changes or improvements to someone else’s code on GitHub.

Merge - combining the suggested changes into the main version of the code.

Branch - a separate version of the code.

Commit - saving your changes in the code's history.

Fork - making your own copy of someone else's code to modify it separately.

Clone - Copying a repository onto your own computer to work on it locally.

3. Actors

3.1.

Name: End user.

Goal: Edit a photo or video.

Responsibilities: Specify the prompt and call the CVIP tool.

3.2.

Name: Project developer.

Goal: Get the updates to the end user and protect their own intellectual property (namely the project).

Responsibilities: They have to provide the end user with the device's firmware (and its updates).

3.3.

Name: Contributor.

Goal: Suggest new features and extensions to the CVIP tool.

Responsibilities: Suggest their code with Pull Request on Github, describing their modifications.

4. Functional requirements

4.1. Use-case <UC-1-1>

Users want to upscale an image by applying a specific upscaling filter.

Actors: Photographers, Graphic Designers.

Goals: User receives the upscaled image in a specific directory.

Precondition: User has access to the CVIP tool and the image to be upscaled.

Extensions: None.

Mains success scenario:

1) User starts forming the call of CVIP with "./CVIP ".

2) User specifies the input image containing faces with "[-i=path/to/input_img_name.format]".

3) User specifies the "Upscale image" filter and desired version of it (Bilinear, Bicubic, Nearest Neighbours).

4) User specifies the desired scaling factor.

5) User specifies the output file name with "[-o=output_img]"

6) User gives a number of parallel processes he wants to use

7) CVIP processes the image, applying the selected upscaling filter.

8) User receives the upscaled image in the same directory with the CVIP tool executable file with the chosen filter applied.

Alternative scenario 1:

1) User starts forming the call of CVIP with "./CVIP ".

2) User specifies the input image containing faces with "[-i=path/to/input_img_name.format]".

3) User specifies the "Upscale image" filter and desired version of it (Bilinear, Bicubic, Nearest Neighbours).

4) User specifies the desired scaling factor.

5) User specifies the output file name with “[-o=output_img]”

6) User gives a number of parallel processes he wants to use

7) CVIP processes the image, applying the selected upscaling filter.

8) User receives the upscaled image in the same directory with the CVIP tool executable file with the chosen filter applied.

9) If the user is not satisfied with the result, they should repeat steps 1-8 until they are satisfied with the result.

Alternative scenario 2:

1) User starts forming the call of CVIP with “./CVIP “.

2) User specifies the input image containing faces with “[-i=path/to/input_img_name.format]”.

3) User specifies the "Upscale image" filter and desired version of it (Bilinear, Bicubic, Nearest Neighbours).

4) User specifies the desired scaling factor.

5) User specifies the output file name with “[-o=output_img]”

6) User gives a number of parallel processes he wants to use

7) CVIP throws an exception because of illegal input, specifying what has gone wrong.

8) User should repeat steps 1-6 with the correct input.

4.2. Use-case <UC-1-2>

User wants to detect faces and blur them.

Actors: Photographers, Privacy-conscious users.

Goals: The user receives the image with blurred faces in a specified directory.

Precondition: The user has access to the CVIP tool and the image in which they want to detect and blur faces.

Extensions: None.

Main Success Scenario:

1) User starts forming the call of CVIP with “./CVIP “.
2) User specifies the input image containing faces with “[-i=path/to/input_img_name.format]”.
3) User specifies the “face_blur” filter.
4) User configures face blur parameters with “face_blur:10”.

5) User specifies the output file name with “[-o=output_img]”
6) User gives a number of parallel processes they want to use..
7) User executes what they formed: “./CVIP [-i=input_img_name.format]face_blur:10[-o=output_img] 4”
8) The CVIP tool processes the image, detecting faces and applying blurring to the identified faces.
9) CVIP generates the image with blurred faces and “output_img” name.
10) User receives the image in the same directory with the CVIP tool executable file with blurred faces in the current directory.

Alternative scenario 1:

1) User starts forming the call of CVIP with “./CVIP “.
2) User specifies the input image containing faces with “[-i=path/to/input_img_name.format]”.
3) User specifies the “face_blur” filter.
4) User configures face blur parameters with “face_blur:10”.
5) User specifies the output file name with “[-o=output_img]”
6) User gives a number of parallel processes they want.
7) User executes what they formed: “./CVIP [-i=input_img_name.format]face_blur:10[-o=output_img] 4”
8) The CVIP tool processes the image, detecting faces and applying blurring to the identified faces.
9) CVIP generates the image with blurred faces and “output_img” name.
10) User receives the image in the same directory with the CVIP tool executable file with blurred faces in the current directory.
11) If the user is not satisfied with the result, they should repeat steps 1-8 until they are satisfied with the result.

Alternative scenario 2:

1) User starts forming the call of CVIP with “./CVIP “.
2) User specifies the input image containing faces with “[-i=path/to/input_img_name.format]”.
3) User specifies the “face_blur” filter.
4) User configures face blur parameters with “face_blur:10”.
5) User specifies the output file name with “[-o=output_img]”
6) User gives a number of parallel processes they want to use.
7) User executes what they formed: “./CVIP [-i=input_img_name.format]face_blur:10[-o=output_img] 4”
8) CVIP throws an exception because of illegal input, specifying what has gone wrong.
9) User should repeat steps 1-6 with the correct input.

4.3. Use-case <UC-2-1>

Contributor wants to add their code modifications to the project.

Actors: Contributor, Project Developers, Users.

Goals: Add their code to the main branch of GitHub Project.

Precondition: Contributor needs to be authorized in GitHub.

Extensions: None.

Main Success Scenario:

1) Contributor forks the CVIP project on GitHub.
2) Contributor clones the repository to their own PC.
3) Contributor adds new features, filters or anything else.
4) Contributor follows the project code-style and adds comments to their code.
5) Contributor commits and pushes changes to their fork of the project.
6) Contributor makes Pull Request to the main branch of the project, describing what they have added.
7) Project Developers review the suggested changes.
8) Project Developers add suggested changes to the main branch of the project.
9) Users update the tool to the latest version.
10) Users can now use new features.

Alternative Scenario 1:

1) Contributor forks the CVIP project on GitHub.
2) Contributor clones the repository to their own PC.
3) Contributor adds new features, filters or anything else.
4) Contributor follows the project code-style and adds comments to their code.
5) Contributor commits and pushes changes to their fork of the project.
6) Contributor makes Pull Request to the main branch of the project, describing what they have added.
7) Project Developers review the suggested changes.
8) Project Developers do not accept suggested changes.
9) Project Developers send feedback why they have declined the Pull Request.
10) Contributor may try to fix the problems described by Project Developers and come back to the step 6.

Alternative Scenario 2:

1) Contributor forks the CVIP project on GitHub.
2) Contributor clones the repository to their own PC.
3) Contributor adds new features, filters or anything else.
4) Contributor follows the project code-style and adds comments to their code.
5) Contributor commits and pushes changes to their fork of the project.
6) Contributor makes Pull Request to the main branch of the project, describing what they have added.
7) Project Developers review the suggested changes.
8) Project Developers accept the idea but suggest some changes.
9) Contributor fixes changes that were suggested and commits new version of the code.
10) Project Developers add suggested changes to the main branch of the project.

11) Users update the tool to the latest version.
12) Users can now use new features.

5. Non-functional requirements

5.1. Environment

5.1.1. Supported Hardware Platforms

Framework is compatible with hardware platforms macOS and Linux.

5.1.2. Programming Languages

Framework is developed using Python.

5.1.3. Libraries

CVIP uses libraries such as OpenCV, numpy, mediapipe, dlib and its dependencies for its core functionality. All dependencies are described in the “requirements.txt” file.

5.2. Performance

5.2.1. Response time

The CVIP call responds to user commands promptly, with an execution time depending on the size of input files.

5.2.2. Throughput

CVIP supports processing images in parallel processes with throughput depending on the complexity of a prompt.

5.2.3. Capacity

The tool handles large files, supporting images and videos with resolutions up to 1920x1080 without significant performance degradation.

5.3. Reliability

5.3.1. Availability

CVIP aims for a high availability rate with an expected uptime of at least 80%.

5.3.2. Failures

The tool is resilient to failures and recovers gracefully from errors. It should minimize the frequency and severity of failures.

5.3.3. Recoverability

In the event of a crash or error, CVIP should provide a clear error message for troubleshooting.

5.4. Extensibility

5.4.1. Extension points

CVIP can be extended by developers or collaborators on GitHub. Extensions can be a GUI, filters for image/video editing or modules that use CV methods.

5.4.2. Scalability

It is possible to run the CVIP tool with multiple parallel calls at the same time on the same device.

5.4.3. Configurability

User is able to specify the number of parallel processes that CVIP uses in runtime.

5.5. License

The CVIP is licensed under the MIT License.