THE DESIGN AND IMPLEMENT OF PEOPLE COUNTING SYSTEM

XIN ZHU

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 HIGH-LEVEL ANALYSIS AND DESIGN

1.1 System Architecture

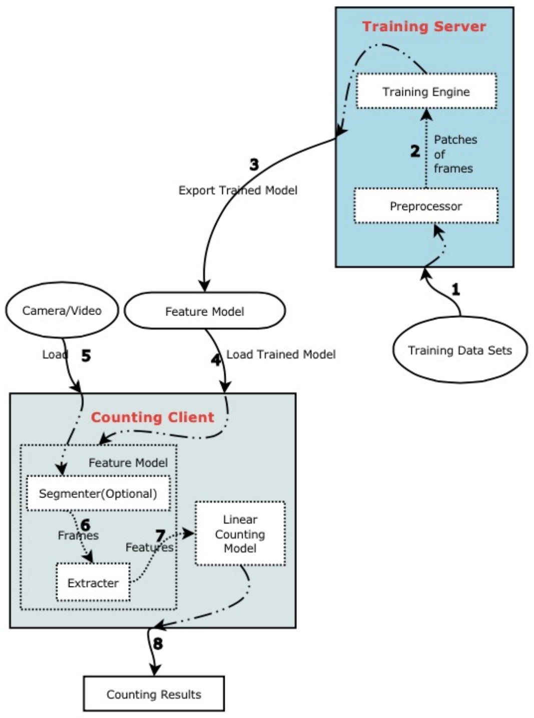


Figure 1: People Counting System Architecture Overview

The people counting system consists of two components that are training server and counting client. They take some different responsibilities respectively. The duty of training server is to train a feature model that will be used in counting client based on training data sets. The customers of counting system usually do not need to care about this one since the feature models generally have been trained by product vendors. The responsibility of counting client is to generate the counting results of each frame from input video or camera.

TRAINING SERVER The training server is the crucial component of counting system that includes two primary modules, which are preprocessor and training engine. Before training a feature extracting model, preprocessor normally involves preprocessing to convert the original training data sets into appropriate format training engine expects. After obtaining the appropriate format of training data sets, preprocessor will commit the preprocessed training data sets with appropriate format such as patch-sized frames to the training engine. While receiving formatted training data sets, the training engine will carry out a series of training process such as convolution, local contrast normalization, pooling and etc. to generate a feature extracting model counting clients will use.

COUNTING CLIENT The counting client is the major operating platform for customers and is also the other important component of people counting

* *Department of Electronic Engineering, The Chinese University of Hong Kong, ShaTin, Hong Kong*

system. The users can employ various functional button embedded in main window of client application to load video or camera to complete the task of crowds analysis.

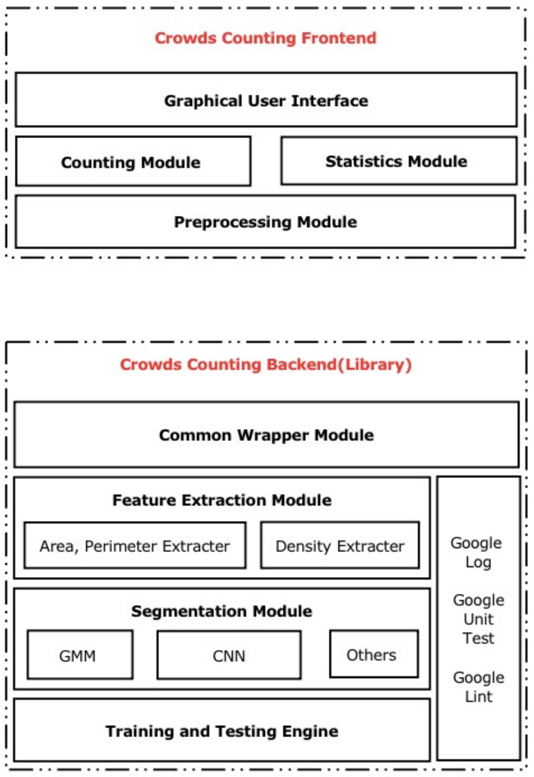


Figure 2: Software Architecture Overview

The figure above shows the critical modules of people counting software system. The whole software system is comprised of crowds counting frontend system and backend system.

FRONTEND SYSTEM As an external window of the whole software system, frontend system is a crucial module. It covers several different submodules. Graphical User Interface of frontend system is the only operating platform for users, it guides how users interact with crowds counting system. counting module employs interfaces of backend system to real time analyze and count the number of people in the video or camera. statistics module adopts modern scientific plotting tool to real time plot a two-dimensional chart with horizontal axis showing the time or frame range and vertical axis showing the number of people. preprocessing module is used as one of initialization modules to prepare the pre-work. These four modules form the frontend system of people counting system.

BACKEND SYSTEM As important as frontend system is, backend system also plays a crucial role in people counting system. One wraps all of the logical actions inside of basic programmable functional interface, common wrapper module. training and testing engine is the most important module in people counting system, training engine is used as trainer to train a model fitting training data sets well, which can be used by tester, however, testing engine is used as tester to use the model to segment out crowds and extract feature from frame. segmentation module and feature extraction module both are based on the engine module and implement a series of functions through the use of interfaces provided by engine module. Meanwhile backend system also employs some useful google tools such as google log and google lint to maintain the whole software system effectively.

1.2 Control and Data Flow

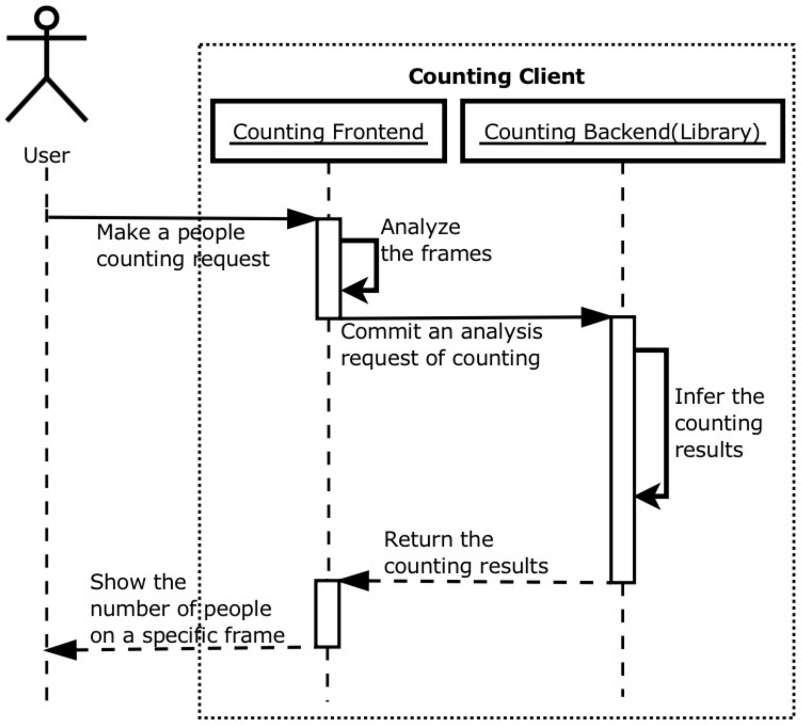

User
Counting Client
Counting Frontend
Counting Backend(Library)
Make a people counting request
Analyze the frames
Commit an analysis request of counting
Infer the counting results
Return the counting results
Show the number of people on a specific frame

Figure 3: Interactive Processing Sequence Diagram

The figure above shows the interactive processing of users and people counting system. solid arrows represents request relationship, dashed arrows represents response relationship.

Users use counting system by operating the GUI window of the frontend system. Users send a people counting request to the fronted system first, the fronted system then preprocesses and commits the request to the backend system, after inferring, the backend system return the counting results to the frontend and users in turn.

2 LOW-LEVEL ANALYSIS AND DESIGN

This part of design extends the details mentioned in the above chapter. It contains core modules, data structure and core algorithm.

2.1 Core Modules

This section discusses several the most important software system modules including segmenter, extracter and engine. segmenter and extracter is the emphasis and core of this section.

2.1.1 *Segmenter*

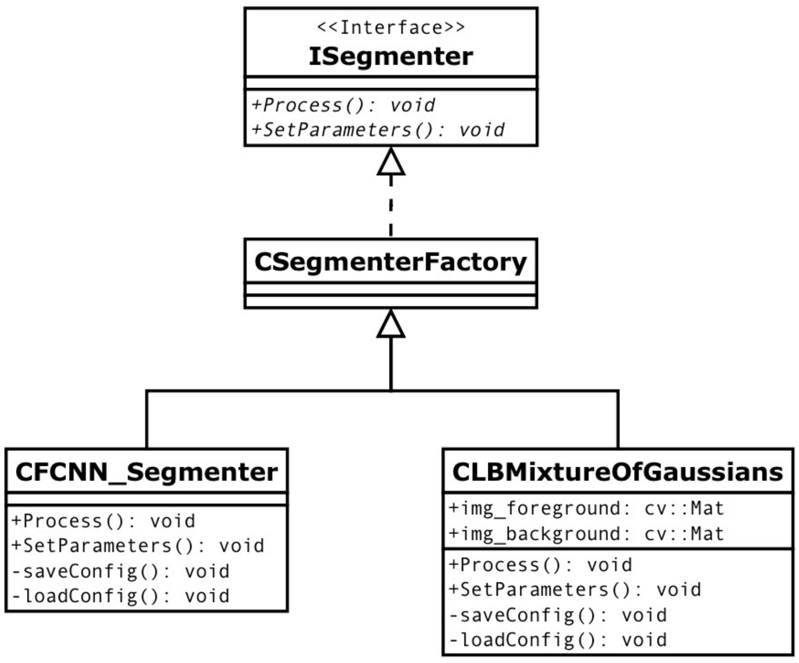


Figure 4: Class Segmenter Overview

The figure depicted above elaborates the segmenter class inheritance and hierarchy structure.

A design model of the factory model is employed to design a segmenter factory that defines a product architecture. FCNN_Segmenter and LBMixtureOfGaussians both are the products from segmenter factory. They inherit from a common base class, ISegmenter, which is a abstract class defining the unified interfaces. These two distinct segmenters both inherit the same interface such as process and setparameters and create the private functions and attributes belong to each other respectively.

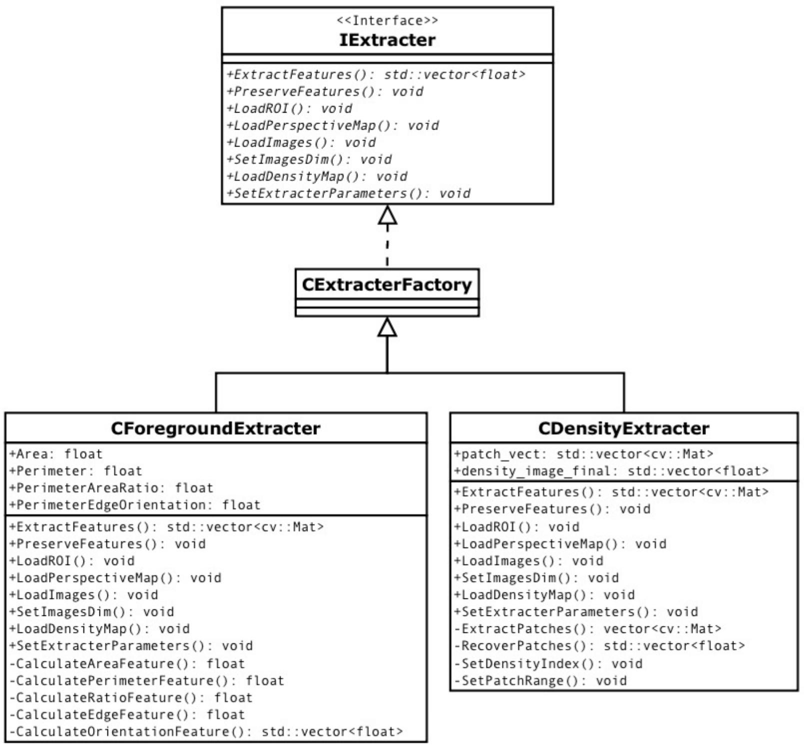
FCNN_Segmenter is a fully convolutional neural network (FCNN) for crowd segmentation. By employing 1 by 1 convolution kernels to substitute the fully connected layers in Convolution Neural Network(CNN), FCNN takes whole images as inputs and directly generates segmentation results by one pass of forward computation. FCNN is actually a model generated by engine with specific input of network structure.

LBMixtureOfGaussians is a traditional segmentation method implemented by adopting the principle of mixture of gaussians. It will not generate a model for the task of segmentation and only employs simple function interfaces to output segmentation map of original frame. This method can not handle the situation that people is stationary.

Currently, our system recommends choosing FCNN_Segmenter as the default segmenter.

2.1.2 *Extracter*

The figure depicted below elaborates the extracter class inheritance and hierarchy structure.



<<Interface>>
IExtracter
+ExtractFeatures(): std::vector<float>
+PreserveFeatures(): void
+LoadROI(): void
+LoadPerspectiveMap(): void
+LoadImages(): void
+SetImagesDim(): void
+LoadDensityMap(): void
+SetExtracterParameters(): void
CExtracterFactory
CForegroundExtracter
+Area: float
+Perimeter: float
+PerimeterAreaRatio: float
+PerimeterEdgeOrientation: float
+ExtractFeatures(): std::vector<cv::Mat>
+PreserveFeatures(): void
+LoadROI(): void
+LoadPerspectiveMap(): void
+LoadImages(): void
+SetImagesDim(): void
+LoadDensityMap(): void
+SetExtracterParameters(): void
-CalculateAreaFeature(): float
-CalculatePerimeterFeature(): float
-CalculateRatioFeature(): float
-CalculateEdgeFeature(): float
-CalculateOrientationFeature(): std::vector<float>
CDensityExtracter
+patch_vect: std::vector<cv::Mat>
+density_image_final: std::vector<float>
+ExtractFeatures(): std::vector<cv::Mat>
+PreserveFeatures(): void
+LoadROI(): void
+LoadPerspectiveMap(): void
+LoadImages(): void
+SetImagesDim(): void
+LoadDensityMap(): void
+SetExtracterParameters(): void
-ExtractPatches(): vector<cv::Mat>
-RecoverPatches(): std::vector<float>
-SetDensityIndex(): void
-SetPatchRange(): void

Figure 5: Class Extracter Overview

A design model of the factory model is employed to design a extracter factory that defines a product architecture. ForegroundExtracter and DensityExtracter both are the products from extracter factory that instantiates an object of the specific type of extracter. They inherit from a common base class, IExtracter, which is a abstract class defining the unified interfaces. These two distinct extracters both inherit the same interface such as ExtractFeatures, PreserveFeatures, LoadROI, LoadPerspectiveMap, setExtracterParameters, etc., and create the private functions and attributes belong to each other respectively.

ForegroundExtracter is a traditional feature extracter that employs common algorithm to extract features of area, perimeter, etc. By combining these distinct features and putting these features as input of linear regression function, a linear model for counting the number of people will be generated. Besides Inheritance from abstract class IExtracter, ForegroundExtracter creates private functions such as CalculateAreaFeature, CalculatePerimeterFeature, etc. to complete requirement of various internal functions.

DensityExtracter is a novel feature extracting method based on convolution neural network with a specific input of network structure describing how to stack layers for extracting the feature of density on each pixel. DensityExtracter is actually a model generated by engine. The model should fit the training data sets well and have enough generalization ability for applying the model to various different scene. Besides Inheritance from abstract class IExtracter, DensityExtracter creates private functions such as ExtractPatches, RecoverPatches, SetPatchRange, SetDensityIndex, etc. to complete requirement of various internal functions.

Our current system recommends choosing DensityExtracter as the default extracter for extracting features more accurate compared with ForegroundExtracter.

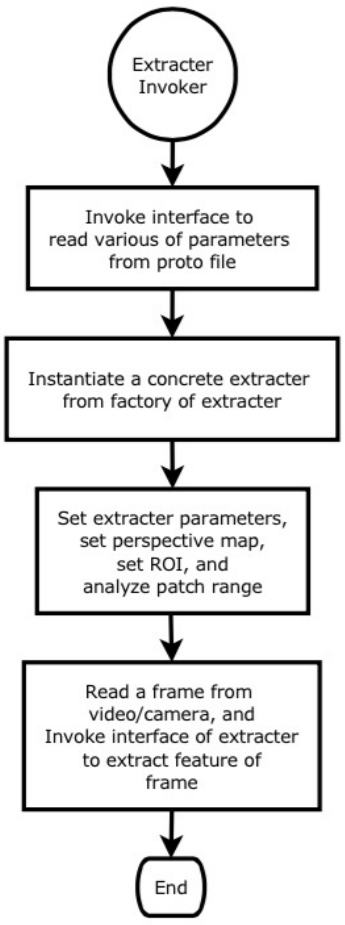

Extracter Invoker
Invoke interface to read various of parameters from proto file
Instantiate a concrete extracter from factory of extracter
Set extracter parameters, set perspective map, set ROI, and analyze patch range
Read a frame from video/camera, and Invoke interface of extracter to extract feature of frame
End

Figure 6: Extracter Invoking Flowchart

The figure depicted above shows the extracter invoking flowchart.

For calling extracter to extract features, invoking interface to read various of parameters from proto file is the first step. After initialization, extracter factory will instantiate a concrete extracter, DensityExtracter. calling interfaces to set parameters, perspective map, ROI, and to analyze the range of extracting patches is the third step. the last step is to read a frame and invoke interface of DensityExtracter to extract feature of frame.

2.1.3 *Engine*

Engine is the most important module in the people counting system and is actually a implementation of Convolution Neural Network(CNN). In our system, FCNN_Segmenter and DensityExtracter both are based on implementation of CNN. We will not give more details about the design of engine in this version.

2.2 Data Structure and Algorithm

This section discusses the most important data structure and algorithm used by system.

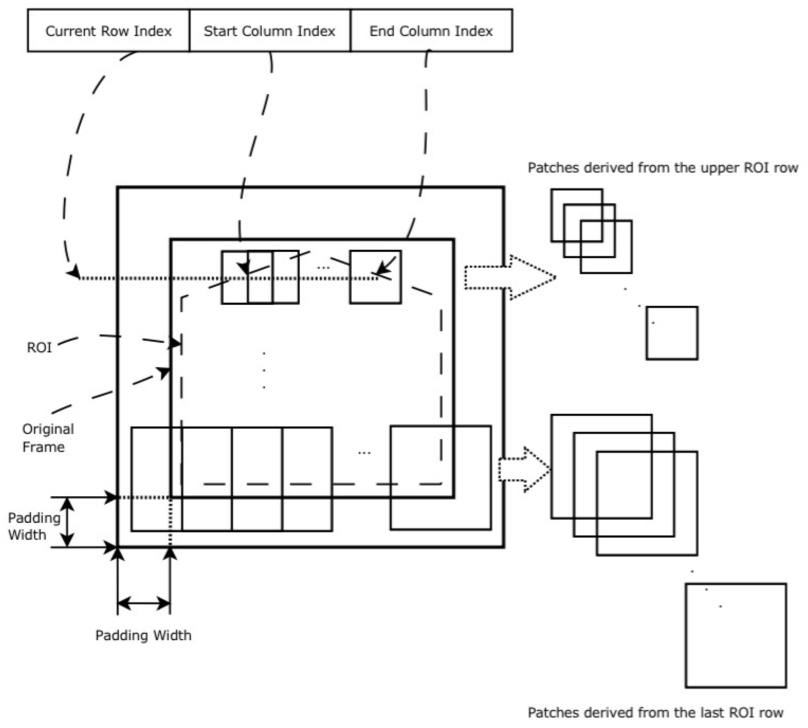


Figure 7: Principle of Extracting Patches

The figure above describes the important principle of extracting patches for preparing input data sets of DensityExtracter.

DATA STRUCTURE patch map is a kind of std::map<key, value> whose key represents the row index on which patches will be extracted and whose value is type of integer vector representing the start column index and end column index on the specific extracting patches row. Hence, patch map is of structure of std::map<int, std::vector<int> >.

PRINCIPLE The next steps give a summary of principle of extracting patches.

FIRST STEP Before extracting patches, calculate the range of extracting patches and store the range to patch map data structure.

SECOND STEP Each of entry of patch map contains the extracting row and extracting start column and end column on the row. locate the position on the frame according to entry information of patch map.

THIRD STEP Dig square patches with size determined by perspective value of current extracting row.

FOURTH STEP Resize all of different size of patches to a fixed size.

The following algorithms give a relative detailed description of principle of extracting patching.

```
Algorithm   Calculating range of extracting patches
Input:
    a. original frame;
    b. ROI;
    c. perspective map;
Output:
    a. patch map containing the range of extracting patches is the structure of
    std::map<key, value> whose key represents the row (stride := perspective_weight) on
    which patches are going to be extracted and whose value is a type of integer vector
    representing the start column and end column of extracting on the specific row.;
 1: generate the start and end row for extracting patches based on the range of ROI;
 2: for each row, stride := the perspective weight of the current row do
 3:     find the first column for extracting patches
 4:     insert index of row to the key section of patch map and insert index of first column
        to the first part of value section of patch map
 5: end for
 6: for each row of patch map, stride := 1 do
 7:     find the last column for extracting patches based on property of ROI
 8:     insert index of last column to the second part of value section of patch map
 9: end for
10: output a patch map containing the range of extracting patches.
```

Figure 8: Calculating range of extracting patches

```
Algorithm   Extracting patches
Input:
    a. patch map containing the range of extracting patches is the structure of
    std::map<key, value> whose key represents the row (stride := perspective_weight) on
    which patches are going to be extracted and whose value is a type of integer vector
    representing the start column and end column of extracting on the specific row.
    b. frame with paddings
    c. perspective map
Output:
    a. fixed size frame patches
 1: let iterator points to the position of the first entry of patch map
 2: for iterator ≤ position of the last entry of patch map, stride := 1 do
 3:     let row := iterator->key
 4:     let start_column := iterator->value[0]
 5:     let end_column := iterator->value[1]
 6:     let perspective_weight := perspective_map[row]
 7:     for start_column ≤ each column ≤ end_column, stride := perspective_weight do
            dig a square patch with 2*perspective_weight width from frame
 8:     end for
 9: end for
10: output a vector containing fixed size patches
```

Figure 9: Extracting Patches Algorithm

```
Algorithm   Recovering patches
Input:
    a. map of number of people on each pixel derived from engine;
Output:
    a. a vector containing the density of people on each pixel;
 1: let iterator points to the position of the first entry of patch map.
 2: for each iterator ≤ position of the last entry of patch map, stride := 1 do
 3:     let row := iterator->key;
 4:     let start_column := iterator->value[0];
 5:     let end_column := iterator->value[1];
 6:     let perspective_weight := perspective_map[row];
 7:     for start_column ≤ each column ≤ end_column, stride := perspective_weight do
            calculate the density of people on each pixel that lays in the square of 2*perspec-
            tive_weight width
 8:     end for
 9: end for
10: generate the density vector in the range of ROI and output the vector.
```

Figure 10: Recovering Patches Algorithm