Devin Doman

OCR Design

See systemUML.png for system dependency structure.

UI System

**DesktopOCR**

Type: Class

Responsibilities:

* Run the OCR program for the desktop user
* Display the user interface

Interface:

* static void main()

Starting point for the desktop application. Calls the initUI() method.

Processing:

* void initUI()

Initialize the components of the user interface: text box to prompt for the file path to the image to process, an OCR button to perform the OCR on the image indicated, a train button to provide training data from the image indicted. Initialize the OCR button with the onOCRClick() method as its onClickListener and the train button with onTrainClick().

* void onOCRClick()

Inner class. Get the file path from the text box. Create a new ImageReader object and call its readImage() method. Call the extractString() method of the OCR object and show the resulting string to the user.

* void onTrainClick()

Inner class. Get the file path from the text box. Create a new ImageReader object and call its readImage() method. Initialize a new Trainer object and call its go() method.

Data: none

**AndroidOCR**

Type: Class

Responsibilities:

* Run the OCR program for the android user
* Display the user interface

Interface:

* static void main()

Starting point for the android application. Calls the initUI() method.

Processing:

* void initUI()

Initialize the components of the user interface: text box to prompt for the file path to the image to process, an OCR button to perform the OCR on the image indicated, a train button to provide training data from the image indicted. Initialize the OCR button with the onOCRClick() method as its onClickListener and the train button with onTrainClick().

* void onOCRClick()

Inner class. Get the file path from the text box. Create a new ImageReader object and call its readImage() method. Call the extractString() method of the OCR object and show the resulting string to the user.

* void onTrainClick()

Inner class. Get the file path from the text box. Create a new ImageReader object and call its readImage() method. Initialize a new Trainer object and call its go() method.

Data: none

**ImageReader**

Type: Class

Responsibilities:

* Reads an image from a given location

Interface:

* ImageReader(File pPath)

Construct an ImageReader object with the given file path.

* Image readImage()

Read the image from the file system and return the contents in an Image object.

Processing: none

Data:

* File mPath: Store the path the image on the file system.

OCR System

**OCR**

Type: Class

Responsibilities:

* Extract characters from an image and convert to a String

Interface:

* static String extractString(Image pImage)

Call the preprocess() method of the Preprocessor object. Loop through each character image returned and create a new FeatureExtractor for it and call the run() method. Call the getFeaturePoint() method of the FeatureExtractor. Initialize a new PatternRecognizer with the feature point passed as a parameter and call the run() method. Call the getCharacter() method of the PatternRecognizer(). Add the character returned to the result String. Return the result String.

Processing: none

Data: none

**Preprocessor**

Type: Class

Responsibilities:

* Convert an image into an array of images containing processed characters
* If connected components exceed distance threshold include a blank

Interface:

* static Collection<Image> preprocess(Image pImage)

Use the methods of the ImageProcessingLibrary to preprocess the image, locate the characters, and return them in an array of Images.

Processing: none

Data: none

**FeatureExtractor**

Type: Class

Responsibilities:

* Convert a character image into a collection of features
* Allow for concurrent processing

Interface:

* FeatureExtractor(Image pCharacter)

Create a new FeatureExtractor object for the character image given.

* Collection<Double> getFeaturePoint()

Returns the feature point calculated by the run() method for the character image given.

* void run()

Implements the Runnable interface. Uses the ImageProcessLibrary methods to normalize the character image, provide a character contour, and a thinned character. Uses the FeatureExtractionLibrary methods to calculate and feature data. The data is then assembled in a specific order to create a feature point. The feature point is set in mFeaturePoint.

Processing: none

Data:

* Image mCharacter: Store the character image
* Image mNormalized: Store the normalized character image
* Image mContour: Store the character contour
* Image mThin: Store the thinned character
* Collection<Double> mFeaturePoint: Store the feature point

**PatternRecognizer**

Type: Class

Responsibilities:

* Convert a feature point into a character

Interface:

* PatternRecognizer(Collection<Double> pFeaturePoint)

Create a new PatternRecognizer for the feature point given. Calls initTrainingData().

* char getCharacter()

Returns the character calculated by the run() method.

* void run()

Implements the Runnable interface. Calls the kNearestNeighbor() method.

Processing:

* void initTrainingData()

Uses the getTrainingData() method of TrainingDataProxy to initialize mTrainingData

* void calcDistances()

Loops through the training data and calculates the distance between the given feature point and the training feature point using the distance() method. The calculated distance is then stored with the character corresponding with the training feature point in mDistances.

* void sortDistances()

Sorts mDistances by distance.

* void kNearestNeighbor(int k)

Calculates the most frequent character from the k nearest neighbors to the feature point using calcDistances() and sortDistances(). Sets the result in mCharacter.

* double distance(Collection<Double> pPointA, Collection<Double> pPointB)

Calculates the Euclidean distance between the two feature points given.

Data:

* Collection<Double> mFeaturePoint: Store the feature point to compare to the training data.
* char mCharacter: Store the calculated character associated with the feature point given.
* Collection<CharacterFeaturePair> mTrainingData: Store the training data.
* Collection<DistanceCharacterPair> mDistances: Store the distance from the feature point to each training point.
* TrainingDataProxy mProxy: Used to obtain the training data.

**ImageProcessingLibrary**

Type: Class

Responsibilities:

* Provide common methods needed to process images

Interface:

* static Image threshold(Image pImage)

Convert to a binary image, separating foreground and background.

* static Image smoothNoise(Image pImage)

Remove salt and pepper noise.

* static Image correctSkew(Image pImage)

Calculate the skew (aka tilt) of the image. Rotate the image to compensate. If the skew is greater than ±20 degrees throw an exception.

* static Image labelConnectedComponents(Image pImage)

Locate and label the connected components.

* static Image normalize(Image pImage)

Translate a character image to have a fixed height and width.

* static Image traceContour(Image pImage)

Trace the contour of a character.

* static Image thin(Image pImage)

Thin the pixels of the character.

* static Image invert(Image pImage)

Flip the foreground and background.

Processing: none

Data: none

**FeatureExtractionLibrary**

Type: Class

Responsibilities:

* Provide common methods needed to extract features
* Feature points must all be the same length and correspond with other points

Interface:

* static double moment(Image pImage)

Calculates the geometric moment of the image.

* static double area(Image pImage)

Calculates the area of the foreground.

* static double height(Image pImage)

Calculates the height of the image.

* static double width(Image pImage)

Calculates the width of the image.

* static double perimeter(Image pImage)

Calculates the perimeter of the foreground.

* static double compactness(double pPerimeter, double pArea)

Calculates the compactness of the foreground.

* static double numberOfHoles(Image pImage)

Calculates the number of holes in the image.

* static void centroid(Image pImage, double pCentroidX, double pCentroidY)

Calculates the centroid location.

* static void featurePoints(Image pImage, double pEndPoints, double pBranchPoints, double pCrossPoints)

Calculates the number of end points, branch points, and cross points.

* static Collection<Double> xAxisHistogram(Image pImage)

Calculates the x-axis histogram.

* static Collection<Double> yAxisHistogram(Image pImage)

Calculates the y-axis histogram.

* static Collection<Double> strokeOrientation(Image pImage)

Calculates the stoke orientation of the foreground.

* static Collection<Double> lines(Image pImage)

Calculates the location of lines in the image, normalized.

* static Collection<Double> circles(Image pImage)

Calculates the location of circles in the image, normalized.

* static Collection<Double> centroidDistance(Image pImage, int pNormal)

Calculates the centroid distance to the foreground pixels, normalized.

* static Collection<Double> curvatureSignature(Image pImage, int pNormal)

Calculates the curvature signature, normalized.

* static Collection<Double> discreteFourierTransform(Collection<Double> pSignature, int pNormal)

Calculates the discrete Fourier transform on a shape signature.

Processing: none

Data: none

**CharacterFeaturePair**

Type: Class

Responsibilities:

* Store relationship between a feature point and a character

Interface:

* Char getCharacter()

Returns the character.

* Collection<Double> getFeaturePoint()

Returns the feature point.

Processing: none

Data:

* char mCharacter : the character
* Collection<Double> mFeaturePoint: the feature point

**DistanceCharacterPair**

Type: Class

Responsibilities:

* Store relationship between a character and its distance from another feature point.

Interface:

* Char getCharacter()

Returns the character.

* double getDistance()

Returns the distance.

Processing: none

Data:

* char mCharacter : the character
* double mDistance: the distance from the character’s feature point and a the feature point of the unknown character.

Training System

**Trainer**

Type: Class

Responsibilities:

* Allow user to supply knowledge base of feature point-character pairs

Interface:

* Trainer(CharacterPrompter pPrompter, Image pImage)

Create a new Trainer object with the CharacterPrompter and training image given.

* void go()

Use the Preprocessor and FeatureExtractor objects to obtain the array of character images and their respective feature points from the training image. Use the promptUser() method of the CharacterPrompter object to display the character image to the user and retrieve a character from the user. Store the character with the feature point using the insertTrainingData() method of the TrainingDataProxy object.

Processing: none

Data:

* CharacterPrompter mPrompter: Used to prompt the user for a character given an image.
* TrainingDataProxy mProxy: Used to insert data into the training data store.
* Image mTrainingImage: The image to train with

**CharacterPrompter**

Type: Interface

Responsibilities:

* Displays an image and prompts for the character

Interface:

* char promptUser(Image pImage)

Display the image to the user and prompt them for a character. Return the character.

Processing: none

Data: none

**DesktopPrompter**

Type: Class

Responsibilities:

* Prompt the desktop user
* Display a GUI

Interface:

* char promptUser(Image pImage)

Display the image to the user and prompt them for a character. Return the character.

Processing: none

Data: none

**AndroidPrompter**

Type: Class

Responsibilities:

* Prompt the android user

Interface:

* char promptUser(Image pImage)

Display the image to the user and prompt them for a character. Return the character.

Processing: none

Data: none

**PromptActivity**

Type: Class

Responsibilities:

* Show the image and prompt for a char
* Return to previous activity when done

Interface:

* onUpdate()

Display the user interface. Retrieve the character.

Processing: none

Data: none

Training Data System

**TrainingDataProxy**

Type: Class

Responsibilities:

* Store and retrieve the training data
* Singleton

Interface:

* insertTrainingData(CharacterFeaturePair pData)

Insert pData into TrainingData.

* Collection<CharacterFeaturePair> getTrainingData()

Get the training data from TrainingData and return it as an array of CharacterFeaturePairs.

Processing: none

Data: none

**TrainingData**

Type: Class

Responsibilities:

* Store the training data
* Serializable

Interface:

* void insert(CharacterFeaturePair pData)

Add pData to mData.

* Collection<CharacterFeaturePair> getData()

Return mData.

Processing: none

Data:

* Collection<CharacterFeaturePair> mData: the training data