Распознавание элементов пользовательского интерфейса на изображениях

Евсеев Олег, 444 группа
Научный руководитель — д. ф.-м. н. Терехов А.Н.

Консультанты:
Сергей Карашевич (JetBrains),
Антон Ялышев (JetBrains)

Автоматическое тестирование UI — что это?

- Позволяет применить принципы test-driven development к разработке пользовательского интерфейса:
  - Автоматическая проверка корректности и качества
  - Сокращается время, затрачиваемое на верификацию
  - Не имея специальных инструментов, трудно отследить постоянно вносящиеся в программный продукт изменения
- Наибольшие успехи — в web (Selenium WebDriver)
- В JetBrains для этого используется GUI Automation Framework
- Задача проверки формы на визуальное соответствие все равно ложится на плечи людей

Цель и задачи

Цель — разработать фреймворк, позволяющий избавить тестировщиков от [части] рутинной работы по визуальной валидации форм (Swing)

Основные задачи:

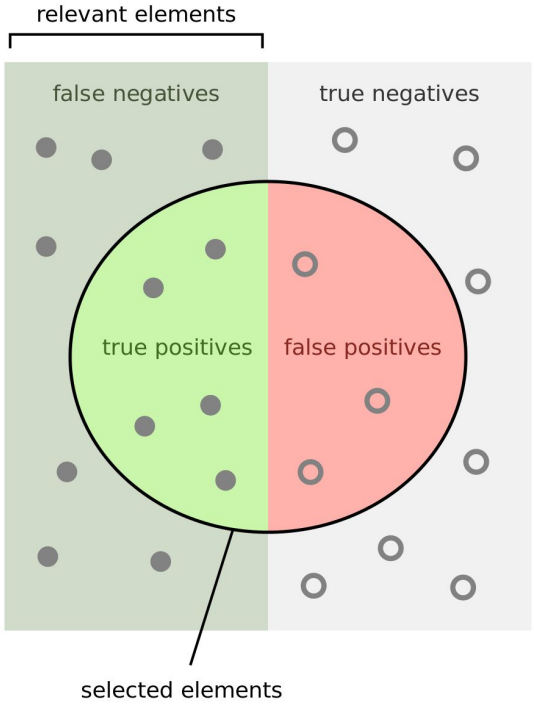
- Научиться сопоставлять кнопки и надписи на скриншоте соответствующим в коде с учетом того, что они могут визуально деформироваться в разумных пределах
- Научиться оценивать качество на «хороших» формах

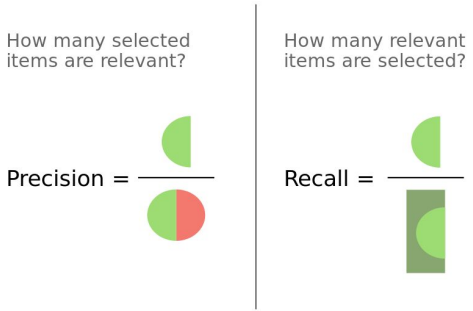
Пока ограничиваемся темной цветовой схемой IDEA (Darcula)

Object detection?

Не совсем:

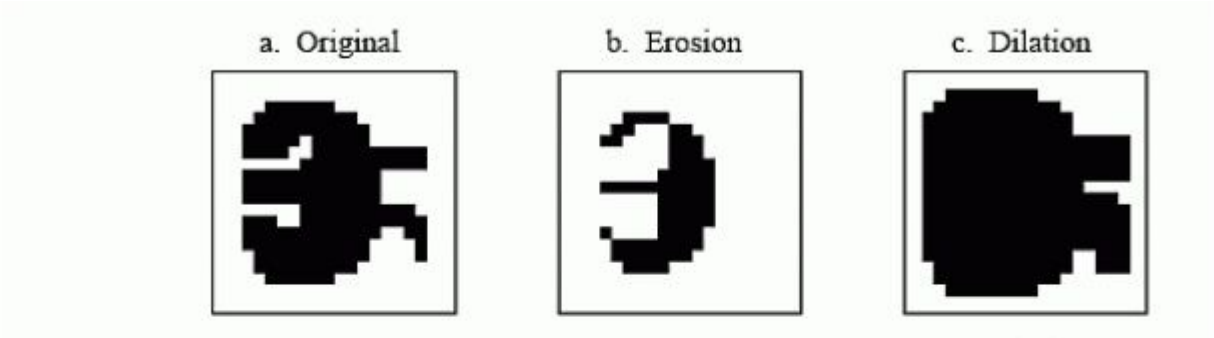
- Точно знаем, что элемент содержится на скриншоте, поэтому достаточно считать только recall (а не, например, mAP)
- Поставленная перед нами задача — получение как можно лучшего (в идеале — выше 90%) recall на формах, которые гарантированно отрисовываются корректно





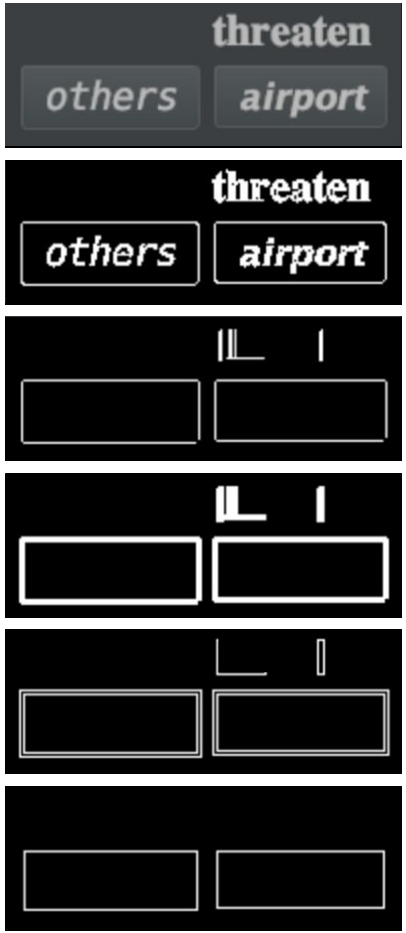
Поиск кнопки на изображении — идеи

- Кнопка — прямоугольник с текстом внутри
- Можно искать прямоугольники обобщенным преобразованием Хафа, но это не оправдано
- Можно использовать морфологию:



Реализация поиска кнопки

- Адаптивная бинаризация либо поиск границ
- Последовательные `erosion/dilation` с ядрами в виде горизонтальной и вертикальной прямой; побитовое ИЛИ двух результатов
- `dilation` для замыкания прямоугольных контуров
- `findContours`
- Отсечение по площади и форме; выбор только внутренних контуров
- Текст внутри прямоугольника распознается Tesseract



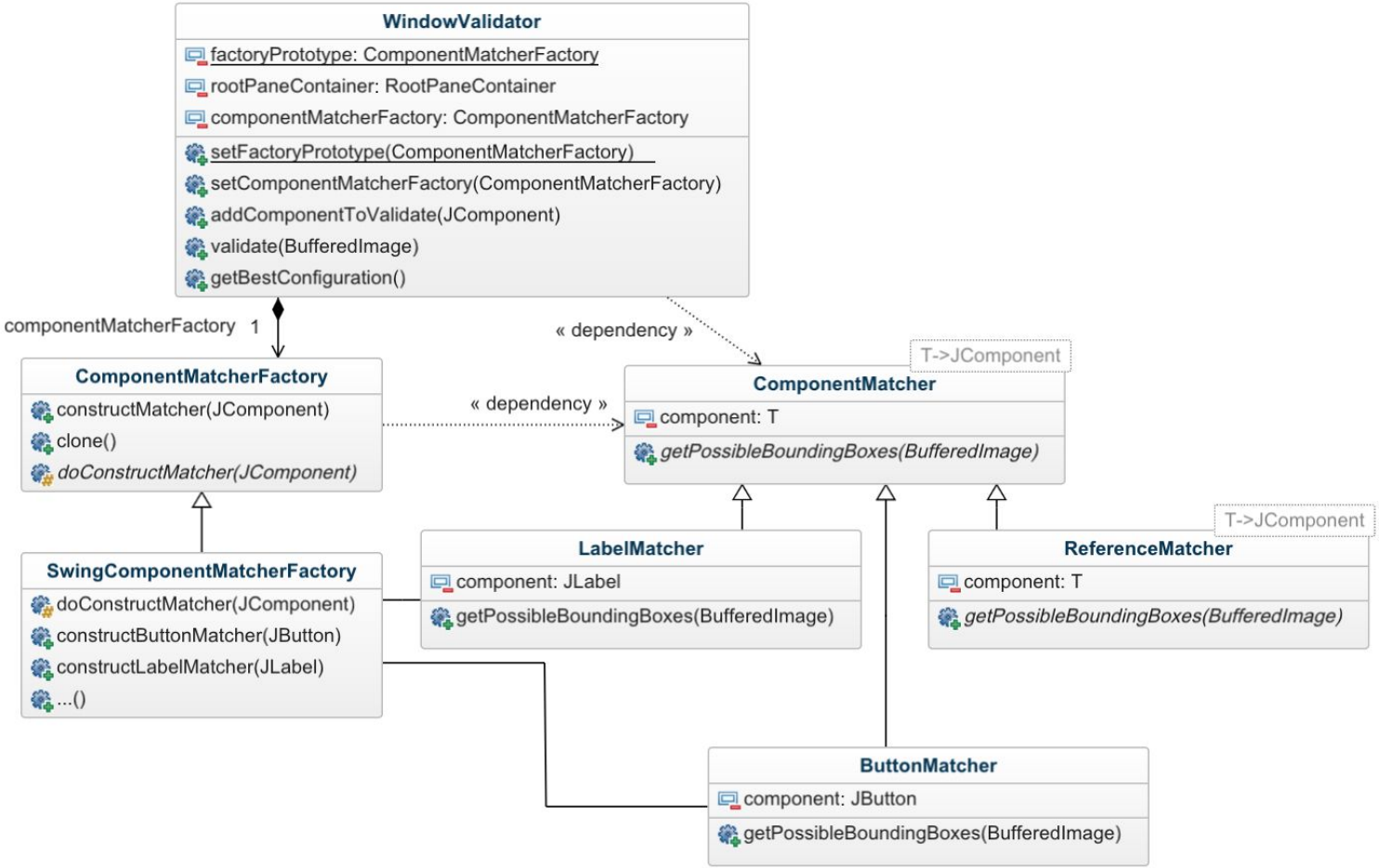
Поиск надписей

- Tesseract умеет сегментировать изображение на уровне символов, слов, строк (на всю ширину изображения!) и абзацев
- Нам нужны блоки текста
- Склеиваем соседние слова при помощи морфологических операций

Алгоритм валидации

- Перебор кандидатов для каждого элемента
- Проверка на то, что прямоугольник еще не использован
- В случае, когда конфигураций несколько, выбирается та, которая максимизирует количество прямоугольников, совпадающих с ограничивающими прямоугольниками, рассчитанными во время исполнения

Элементы, которые пока не имеют специализированного подхода для поиска их на изображении, можем искать при помощи свертки с шаблоном и подсчета кросс-корреляции


WindowValidator
factoryPrototype: ComponentMatcherFactory
rootPaneContainer: RootPaneContainer
componentMatcherFactory: ComponentMatcherFactory
setFactoryPrototype(ComponentMatcherFactory)
setComponentMatcherFactory(ComponentMatcherFactory)
addComponentToValidate(JComponent)
validate(BufferedImage)
getBestConfiguration()
componentMatcherFactory 1
« dependency »
ComponentMatcherFactory
constructMatcher(JComponent)
clone()
doConstructMatcher(JComponent)
« dependency »
T->JComponent
ComponentMatcher
component: T
getPossibleBoundingBoxes(BufferedImage)
SwingComponentMatcherFactory
doConstructMatcher(JComponent)
constructButtonMatcher(JButton)
constructLabelMatcher(JLabel)
...()
LabelMatcher
component: JLabel
getPossibleBoundingBoxes(BufferedImage)
T->JComponent
ReferenceMatcher
component: T
getPossibleBoundingBoxes(BufferedImage)
ButtonMatcher
component: JButton
getPossibleBoundingBoxes(BufferedImage)

Случайные тесты

- Генерируется случайное дерево
- На основе дерева строится форма
  - Листы — компоненты управления
  - Нетерминальные вершины — JPanel
  - Не создаем JPanel, если у вершины один ребенок
- Текст — одно из 3000 наиболее общеупотребимых слов английского языка
- Вызываем `pack()` на такой форме, чтобы все отрисовывалось корректно

Результаты

- Средний recall на 50 наборах по 20 случайно сгенерированных форм (дерево до 20 вершин):

Без увеличения разрешения	С увеличением разрешения	С векторной отрисовкой
72.6±4.0%	84.6±4.4%	85.9±4.5%

- Ряд простых самодельных форм в качестве юнит-тестов; проверяем корректность работы на них перед запуском случайных тестов
- В среднем обработка одной формы занимает 2.6 секунды (Core i5-5257U, dual-core, 2.7 GHz)
- Были проведены попытки дообучить Tesseract, но по причине отсутствия оборудования и датасета качество не получилось сильно улучшить

Выводы

- Получили минимальный рабочий прототип
- Полученные подходы пока покрывают достаточно небольшой набор элементов управления
- Качество распознавания пока недостаточно для применения в реальных условиях — ~15% элементов управления все равно придется просматривать вручную
- Улучшение качества распознавания упирается в обучение Tesseract

Что удалось сделать?

1. Реализация распознавания на скриншоте кнопок при помощи методов компьютерного зрения (с использованием OpenCV) и Tesseract.
2. Реализация распознавания на скриншоте надписей.
3. Реализация алгоритма валидации.
4. Реализация генератора случайных форм для оценки качества распознавания.
5. Реализованные подходы оформлены в расширяемую архитектуру.
6. Связка IDEA + OpenCV + Tesseract + данный фреймворк оформлены в виде Docker-контейнера.
7. Предприняты попытки дообучить Tesseract.