Layout-aware Information Extraction from Semi-structured Medical Images

Kangqi Luo[a], Jinyi Lu[a], Kenny Q. Zhu[a,*], Weiguo Gao[b], Jia Wei[b,*], Meizhuo Zhang[b]

[a]*Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 200240, P.R. China*
[b]*AstraZeneca China, 199 Liangjing Road, Shanghai 201203, P.R. China*

Abstract

Textual information embedded in the medical image contains rich structured knowledge of the particular patient. This paper aims at extracting structured texutal information from semi-structured medical images. Given the recognized text spans of an image preprocessed by optical character recognition (OCR), due to the spatial discontinuity of texts spans as well as potential errors brought by OCR, the structured information extraction becomes more challenging. In this paper, we propose a domain-specific language, called ODL, which allows users to describe the value and layout ~~format~~ of text data contained in the images. Based on the value and spatial constraints described in ODL, the ODL parser associates values found in the image with the data structure in the ODL descrip while conforming to the aforementioned constraints. We conduct experiments on real medical image datasets. DL parser consistently outperforms existing approaches in terms of extraction accuracy, which shows the better tolerance of incorrectly recognized texts, and positional variances between images. This accuracy can be further improved by learning from a few manual corrections.

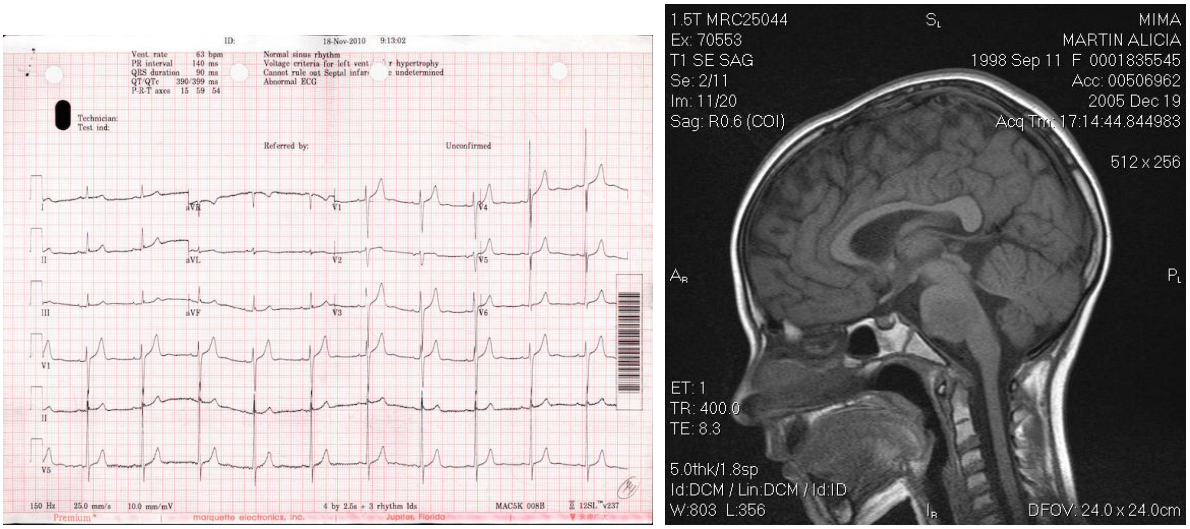
Keywords: information extraction, medical images, electronic medical records, domain-specific language, spatial layout, optical character recognition

1. Introduction

Information extraction is the task of automatically extracting information or knowledge from unstructured or semi-structured documents. In the domain of image processing, the task of textual information extraction (TIE) is automatically detecting and recognizing texts from given images. TIE is applied in a large variety of image categories, such as printed books, newspapers, digital drawings, or even more general images [1]. Optical character recognition (OCR) is the popular approach to solve TIE, turning images of printed text into machine encoded texts. OCR is a hot research topic in recent years, and the rmance is promising on plain-text-based images as novels and reports. For example, Tesseract [2], one of the most popular open source multilingual recognizers, achieved an error rate of 3.72% for recognizing English words and 3.77% for simplified Chinese characters[3]. Based on the exclusive use and high accuracy of OCR, the Google Books [4] and Gutenberg Project [5] have scanned a large number of printed books and converted into text for free and open access.

Due to ever evolving hardware and software, many medical images such as electro-cardio graphs (ECGs), magnetic resonance imaging (MRI), X-ray or ultrasound images irectly printed and stored in hard copy formats. cal images shown in Figure 1 contain a mix of graphics and text, which include technical settings of the hardware used, test measurements and simple diagnoses. In order to build the manageable electronic medical records for patients, there has been a growing demand for extracting such text-based medical information from these medical images. Given the ECG image in Figure 2 as the running example, existing OCR softwares are able to produce texts with spatial layout information, indicating the positions of each recognized text in the image. Figure 3 shows the raw OCR result produced by Tesseract, where recognized texts are organized in the form of XML hierarchy of spans, and we call them "text boxes" throughout this paper. Texts are stored only in leaf text boxes, and the layout information of each box is represented by the coordinates (left, top, right, bottom) of its bounding box. We use the term "bounding box" and "zone" interchangeably in

*Corresponding author.
Email addresses: kzhu@cs.sjtu.edu.cn (Kenny Q. Zhu), jenny.wei@astrazeneca.com (Jia Wei)



(a) ECG (b) MRI

Figure 1: Examples of medical images with textual information.

this paper. It's worth mentioning that OCR softwares may generate incorrect texts. for example, the original text "Vent. rate" in the image is incorrectly recognized as "Vcnt. rule", also "63 bpm" is recognized as "53 bpm".

Apart from scanning the images into digital formats, extracting the structured textual knowledge is more important for building electronic medical records. For example in Figure 2, we would like to extract the attribute-value pairs (e.g., *Vent. rate = 63 bpm*) and possibly other values such as date (e.g., *18-Nov-2010*) and time (e.g., *9:13:02*), since those values endow us with information of the particular patient. Since discovering structured knowledge is beyond the scope of the OCR techniques, the goal of this work is to provide a systematic solution, which enables users to easily specify the data of interest in the images, and automatically extract the corresponding textual values from raw OCR results.

There are some naive solutions for this information extraction task. The first approach is to write regex expressions for each data to be extracted, and apply them to different text boxes of the OCR result. For example, using *"Vent\. rate .* bpm"* to capture the target bpm value. However, this approach suffers from two major problems. First and obviously, incorrect recognized texts lead to mismatches of regex rules. Second, the hierarchical layout of OCR results are not always organized in a human-readable manner. For example in Figure 3, the text "Vcnt. rule" and "53 bpm" are not automatically combined into the same text box, but are rather far apart. In fact, the hierarchical layout is sensitive to many factors, including color, contrast, accidental spots on the prints, or the angle of the scanner camera. In this case, text-based regex rules are not well suited for medical images. Besides, writing regex rules is too ad-hoc and non-trivial for end users.

The second approach is more straightforward: the user annotates all the target zones of each desired data, and simply conducts OCR on each fragment of images. Though intuitive and easy to implement, this approach highly relies on the the accuracy of carefully annotated zones, either too larger or too smaller will affect the local OCR results. In addition, there exists slight positional variations of target zones between two images, even if they share the same format. Therefore, the user have to annotate zones for every individual image, which is tedious and labour intensive.

Another alternative solution involves the page layout analysis technique [6], which includes identifying and organizing the textual regions in the scanned image of a text document. Figure 4 shows the page layout result of the running example. In particular, the technique first segments textual zones (blue blocks) from non-textual zones and arrange them in their original order, then detects individual text unit (red lines under texts) in each zone. Page layout analysis is mainly used for analyzing the semantics of text zones for plain text image documents, and it encounters two problems when applied to this task. First, it is based on the strong assumption that images of the same format share the same structure of

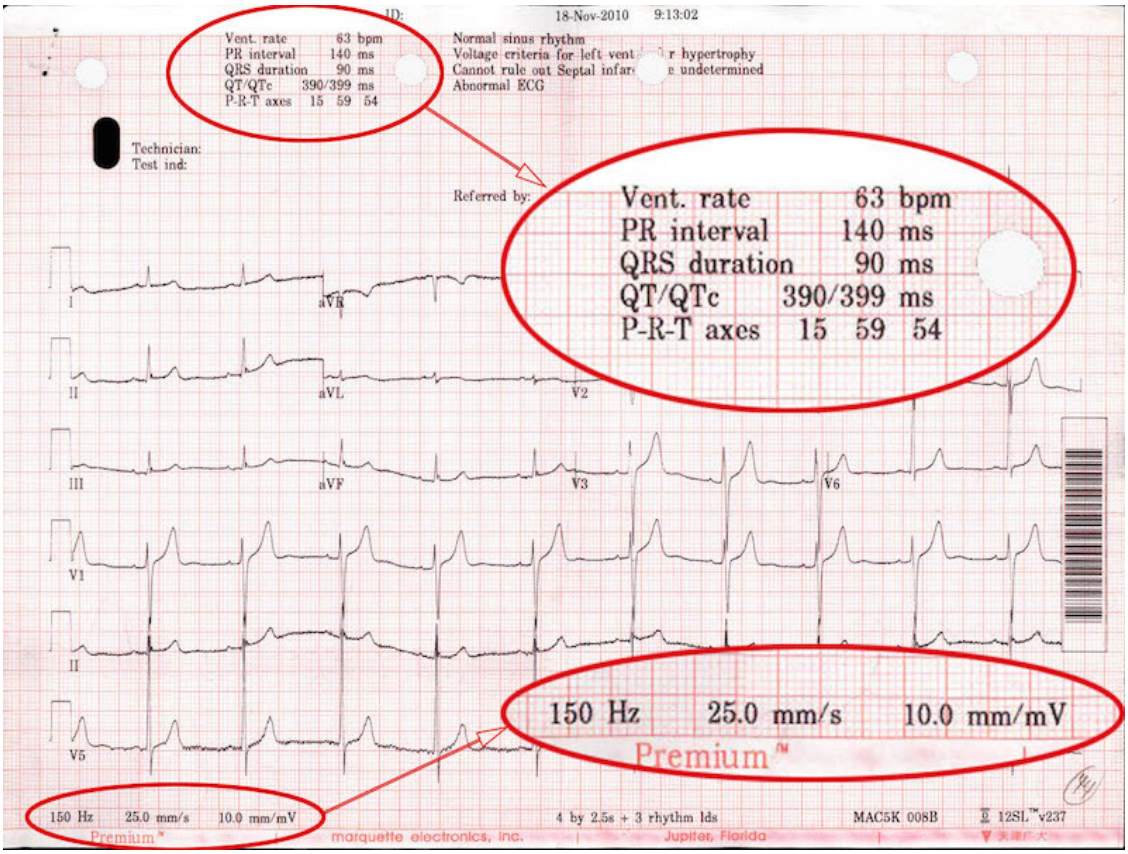


Figure 2: An ECG with interesting text areas marked by red ovals.

page layout result. Noises in the image will affect the page layout result and eventually generate totally error textual information. Second, users must implement an addition wrapper to describe the location (which texts in which regions) of each desired data. This step highly depends on the detail result of page layout, and writing such wrapper is almost intractable for common users without expert knowledge.

In order to attack the limitations of the above approaches, We propose a domain-specific language for describing and extracting structured textual information from the raw OCR data of medical images. We call it OCR description language, or ODL in short. The ODL parser then parses the raw OCR data of the medical images according to the description, and extracts structured textual data in a tree shape. The ODL based solution has three major advantages. First, compared with the previous methods, ODL is more user-friendly. Borrowing the syntax from PADS [7], an ad-hoc data processing language, ODL provides a concrete syntax which allows users to easily define fixed strings and variables to be extracted, customize compositions for better organizing structured data, apply value constraints on variables, and rough spatial constraints on structures. Second, the syntax of ODL is layout-aware. Besides the explicit description of the bounding boxes of individual data, ODL also implies the relative layout between different data elements, which is based on the left-to-right and top-to-bottom data description manner. Those rich layout information support the effectiveness of the ODL parser. Third, the parsing process of ODL is robust. Intuitively, the process aims at finding the best alignments between the data defined in ODL and the text boxes of the raw OCR result, satisfying both value constraints, spatial constraints and their relative layouts. The fuzzy matching strategy is applied to tolerate or even automatically correct recognition errors brought by OCR, as well as slight layout variances between images. Therefore, neither carefully annotated zones nor perfect OCR recognitions are required in the extraction step.

In summary, this paper makes three main contributions.

1. We design a declarative spatial data description

```
<p class="ocr_par" title="box 263 33 444 119">
   <span class="ocr_l" title="box 264 33 336 45">
       <span class="ocrx_w" title="box 264 33 299 45">Vcnt.</span>
       <span class="ocrx_w" title="box 308 34 336 45">rule </span>
   </span>
   <span class='ocr_l'>
       <span class="ocrx_w" title="box 264 51 283 64">PR</span>
       <span class="ocrx_w" title="box 291 51 346 64">Interval </span>
       <span class="ocrx_w" title="box 389 52 411 64">140</span>
       <span class="ocrx_w" title="box 420 55 439 64">ms</span>
   </span>
   ...
   </span>
</p>
<p class="ocr_p" dir="ltr">
   <span class="ocr_l">
       <span class="ocrx_w" title="box 396 33 411 45">53</span>
       <span class="ocrx_w" title="box 420 33 449 48">bpm</span>
   </span>
</p>
```

Figure 3: A fragment of raw OCR results for ECG with layout information.

language for describing both spatial and value constraints in medical images, which can be used to automatically generate parsers for structured information extraction from these images. The syntax of ODL can be generalized to the other image domains (Section 3);

2. We propose a robust ODL parser, which builds the association between the text boxes from raw OCR results and the corresponding description in ODL. During the parsing phase, the parser is able to tolerate the noises and errors brought by OCR recognition, as well as inaccurate bounding boxes of input description (Section 4);
3. We conduct preliminary experimental studies of structured information extraction on real ECG dataset. The end-to-end evaluation result shows that our ODL based solution consistently outperforms existing approaches. Besides, the extraction accuracy further increases by 2%, given only a few number of manual corrections. (Section 5).

2. Approach Overview

In this section, we describe the general framework of our system to extract the structured textual information from the medical images. A running example will be given to detail the different parts of the system.

2.1. Framework

Figure 5 shows the framework of the whole system. The system takes as input one or more medical images of the same format (e.g., the same medical test conducted by the same equipment). All images are processed by the OCR engine, turning into raw OCR results in a structured format such as XML. For each raw OCR result, we ignore its hierarchy and collect all the leaf text boxes as the OCR data, denoted by D. As defined in Equation (1), each leaf text box d is a 5-tuple, which contains the left, top, right, bottom coordinates with the recognized text respectively.

$$D = \{d_1, ..., d_n\}, \quad d = (x_0, y_0, x_1, y_1, text). \tag{1}$$

The other part of the input is the data description of those images, written in ODL. In a practical system, a user can annotate the data description through a graphical user interface, which allows defining strings and variables, adding data constraints and drawing *rough* bounding boxes for different elements in the image. Afterwards, the corresponding ODL description is automatically generated.

The parsing process is the main part of our system. Given the OCR data D and user-defined ODL, we propose the ODL parser which matches all the elements in the provided description with the text from D, and produces parsing results. The parsing results are in the form of a parsing tree, where every leaf node contains both the element of ODL and the corresponding text information. The parser tolerates the errors of OCR recognition and slight layout variances during the extracting process, therefore it's able to produce multiple candidate parsing trees with slight mismatches, and the most suitable result will be selected using scoring functions.

In addition, the system contains an automatic correc-

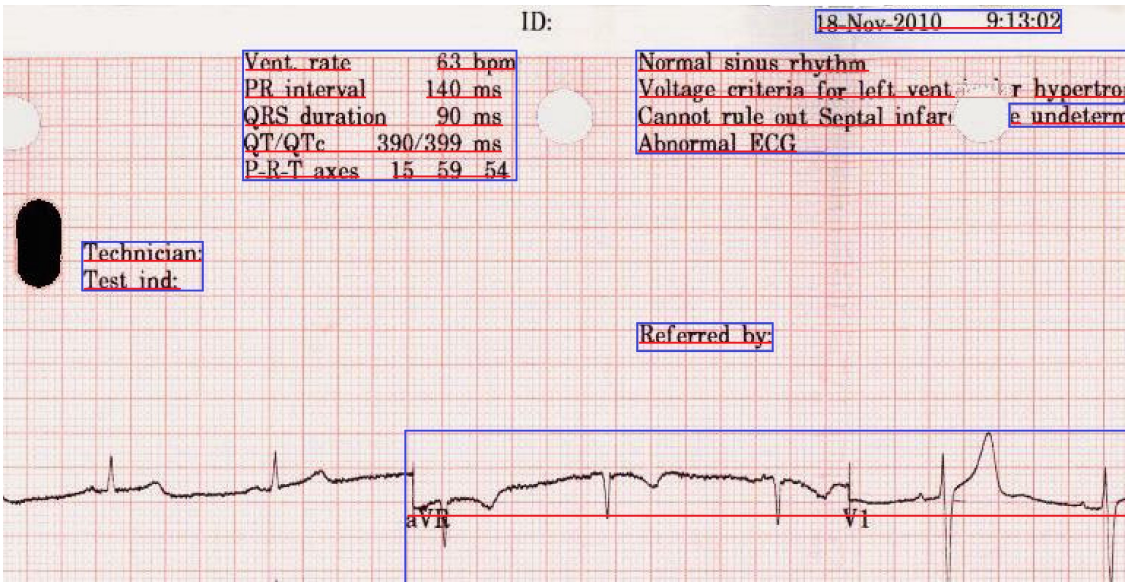


Figure 4: Example result of page layout analysis.

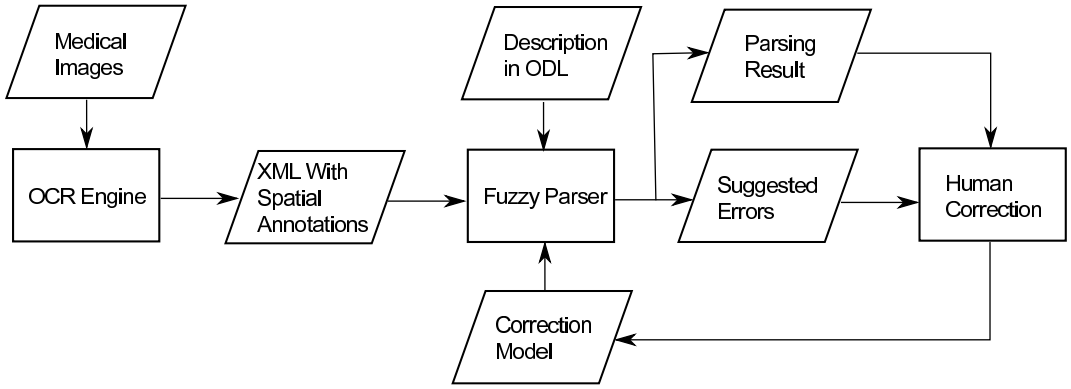


Figure 5: System Framework

tion module, which learns OCR recognition errors and tries to correct errors during the parsing phase. After parsing multiple images, the system collects all detected parsing errors, then provide the user with most frequent parsing errors and prompt the user for possible corrections. Such corrections would then be incorporated into a correction model, which guides the parser to make possible corrections during the subsequent parsing process, and results in fewer errors.

2.2. *Running Example*

We continue using the ECG in Figure 2 as the running example. We are interested in extracting the time, measurements and some basic diagnoses, all highlighted in the red ovals. Fragments of the XML results produced by the OCR software are shown in Figure 3.

There are numerous ways to describe the image using ODL. One of the descriptions is shown in Figure 6, written in the surface syntax. In the description, the simplest expressions are fixed constants and primitive variables. For example, *"Vent. rate"* is a string constant, and *num* is the name of an integer variable ranging from 1 to 12. Users can customize compound types for representing structured textual information. For instance, *time_t* is a *struct* type representing day, month and year information; *month_str* is a *union* type representing enumerable month abbreviations. Users can further specify rough bounding boxes for different expressions. As what depicted in the description, the variable *time* has a rough zone, indicating the corresponding data resides in the top 30% area of the whole image. The keyword *source* stands for the root expression of the entire data to be ex-

```
union month_str{
    "Jan"; "Feb"; "Mar";
    "Apr"; "May"; "Jun";
    "Jul"; "Aug"; "Sept";
    "Oct"; "Nov"; "Dec";
};
union month_t{
    int(1,12)  num;
    month_str  str;
};
struct time_t{
    int(1,31)  day;
    "-";
    month_t month;
    "-";
    int()      year;
};
struct triple_t{
    "Vent. rate";
    hskip(\t)  skip;
    int(60,100)  vr;
    "bpm";
};
```

```
union interpretation_t{
    "Normal ECG";
    "Abnormal ECG";
};
struct parameter_t{
    int()  p1;
    "Hz";
    float(3, 1)    p2;
    "mm/s";
    float(3, 1)    p3;
    "mm/mV";
};
source struct entry_t{
    time_t(<0.0w,0.0h,1.0w,0.3h>)  time;
    triple_t(<0.1w,0.0h,0.5w,1.0h>)  tri;
    interpretation_t(<0.3w,0.0h,1.0w,0.5h>)  inter;
    vskip(\n)[]  skipline;
    parameter_t  para;
};
```

Figure 6: The ODL description in surface syntax.

tracted. In addition, the corresponding abstract syntax of this description is shown in Figure 7, and our later discussion on ODL will use the **abstract syntax** as instead.

The example parsing tree is shown in Figure 8, where leaf nodes are either (constant, value) or (variable, value) pairs. The node *entry_t* serves as the root node of the whole tree, and leaf nodes are marked with "Error" if an imperfect alignment is detected, either because the constant and value doesn't exactly match, or because the value doesn't satisfy the corresponding value constraint. The system will prompt frequent parsing errors for manual corrections. For example, the system detects an error associated with variable $p1$, as "150" is incorrectly recognized as "15o" by the OCR. If the user corrects "15o" into "150", the system learns there is a possibility for the character "0" to be misrecognized for "o" in the current image format. Subsequently, for other images of the same format, the system is able to automatically correct similar errors, e.g., the value for variable $p3$ from "1o.o" to "10.0."

3. Syntax of OCR Description Language

In this section, we introduce the OCR description language, and explain what kind of structured data that ODL is able to describe. The abstract syntax of ODL is formally described in Figure 9. ODL is able to describe both structured data and spatial layouts of textual information within the medical image. In the following parts, we will discuss the ODL syntax and its type system in more detail.

3.1. Primitive Expressions

According to the abstract syntax, we start from the most primitive expressions that ODL can describe: *constant* and *variable*. Constants represent fixed-valued strings or numerical values to be recognized from the image. For example in Figure 7, "Vent. rate" and "mm/mV" are constant expressions, as they always occur in all ECGs of the same format. On the other hand, primitive variables represent numerical (int or float) values varied in different images. In Figure 7, the variable *vr* represents the numerical value of "Vent. rate".It's worth mentioning that, the reason we specially define constants in ODL is to enrich the relative layout information between different expressions, so that the parser can locate the variables in the image more accurately.

3.2. Spatial Expressions

Spatial description expressions include *hskip len* and *vskip len*. As shown in Figure 2, there has a large horizontal margin between "Vent. rate" and "63 bpm", hence we can use *hskip len* to explicitly and approximately describe such horizontal margin between the previous and next expressions. The size of margin is determined by *len*, which has two parts: the length value and its measurement unit. Units can be the absolute pixel, or what are more encouraged, the percentage of

```
{
    {
        day(int, 1, 31),
        "-",
        {
            num(int, 1, 12) |
            {
                "Jan" | "Feb" | "Mar" |
                "Apr" | "May" | "Jun" |
                "Jul" | "Aug" | "Sept" |
                "Oct" | "Nov" | "Dec"
            } as str
        } as month,
        "-",
        year(int)
    }(<0.0w,0.0h,1.0w,0.3h>) as time,
    {
        "Vent. rate",
        hskip \t,
        vr(int, 60, 100),
        "bpm",
    }(<0.1w,0.0h,0.5w,1.0h>) as triple,
    {
        "Normal ECG" |
        "Abnormal ECG"
    }(<0.3w,0.0h,1.0w,0.5h>) as inter,
    vskip \n list as skipline,
    {
        p1(int),
        "Hz",
        p2(float, 3, 1),
        "mm/s",
        p3(float, 3, 1),
        "mm/mV"
    } as para
} as entry_t
```

Figure 7: The ODL description in abstract syntax.

```
entry_t
|-time
  |-(day, "18")
  |-("-", "-")
  |-month
    |-str
      |-("Nov", "Nov")
  |-("-", "-")
  |-(year, "2010")
|-triple
  |-("Vent. rate", "Vcnt. rule", Error)
  |-(vr, "53", Error)
  |-("bpm", "bpm")
|-inter
  |-("Abnormal ECG", "Abnormal ECG")
|-para
  |-(p1, "15o", Error)
  |-("Hz", "Hz")
  |-(p2, "25.0")
  |-("mm/s", "mm/s")
  |-(p3, "1o.o", Error)
  |-("mm/mV", "mm/mV")
```

Figure 8: The example parsing tree of the running ECG.

$$
\begin{aligned}
\text{num} &::= int | float \\
\text{len} &::= num(pixel|l|w) \mid \backslash s \mid \backslash n \\
\text{coor} &::= \langle len_1, len_2, len_3, len_4 \rangle \\
\text{bop} &::= + \mid - \mid * \mid / \mid \% \\
&\quad \mid = \mid ! = \mid < \mid > \mid <= \mid >= \\
\text{c} &::= () \mid int \mid float \mid string \\
\text{e} &::= c && \text{(constant)} \\
&\quad \mid x && \text{(variable)} \\
&\quad \mid hskip\ len && \text{(horizontal skip)} \\
&\quad \mid vskip\ len && \text{(vertical skip)} \\
&\quad \mid \{e_1 | e_2 | ... | e_n\} && \text{(union)} \\
&\quad \mid \{e_1, ..., e_n\} && \text{(struct)} \\
&\quad \mid e\ list && \text{(list)} \\
&\quad \mid e\ as\ x && \text{(binding)} \\
&\quad \mid e_1\ bop\ e_2 && \text{(binary operation)} \\
&\quad \mid e_0(e_1, e_2, ..., e_n) && \text{(constraint)}
\end{aligned}
$$

Figure 9: Syntax of OCR description language.

width (w) or height (h) of the image. Besides, \t stands for a special margin size, which equals to the average width of 4 Latin characters. We can easily estimate this value via the width of each text box in the raw OCR results. Similarly, *vskip len* explicitly describes the vertical margin between expressions, and \n stands for the average height of one Latin character.

3.3. Composition

Compositions are compound expressions constructed from other expressions. These include *union*, *struct*, *list*, *binding*, *bop* and *constraint*.

The first three compositions define more structured and complex type expressions. The *union* expression means there exists multiple potential data or spatial expressions. For example, the union description of *month_str* is the enumeration of all abbreviations of different months. The *struct* expression is used to describe an expression with multiple sub-expressions. All sub-expressions must be described sequentially, following the left-to-right then top-to-bottom manner. As another example in Figure 6, the struct description of *triple_t* contains the constant attribute name, spacing, variable value and constant unit listed from left to right. The *list*

expression indicates that a sequence of similar data or the same spatial expressions should be applied multiple times. In the example, *vskip \n list* represents several blank lines between the data of interpretation and parameters in the images.

The function of *binding* is to give a variable name x to the composition expression e, so that each expression has an identifier in the output parsing tree. The function of *bop* supports basic binary operations between numerical values in the ODL. These two expressions are designed to simplify the description of ODL.

Finally, the *constraint* expression in ODL consists of two categories: value constraints and spatial constraints. Value constraints can be applied to the primitive variable, indicating its type and value range, if the user knows in prior. Since variables are usually numerical, two value constraints are available: $x(int, v_{min}, v_{max})$ and $x(float, length, precision, v_{min}, v_{max})$. For example in Figure 7, $day(int, 1, 31)$ constrains the variable to be an integer ranging from 1 to 31; $p2(float, 3, 1)$ constrains the variable to be a floating number in length 3 and precision 1, without range limits. Spatial constraints have the form $e(coor)$, which can be applied to any expression, and restrict the areas that corresponding data resides in the image. Such positions in ODL are represented by *coor*, the 4-tuple of left, top, right, bottom coordinates. As shown in Figure 7, spatial constraints are applied to several structs: *time*, *tri* and *inter*. These constraints are rather rough and large, as users are encouraged to give larger spatial constraints if they are not so sure of the exact bounding boxes.

3.4. Type System

The inductive typing rules of ODL is shown in Figure 10. *T-VARIABLE* indicates that the type of *variable* is based on the type of name in the typing context. *T-INT ARITH*, *T-INT REL*, *T-FLOAT ARITH* and *T-FLOAT REL* indicate that the two expressions of the *bop* are of the same type (int or float), and the final type of the *bop* expression is based on the binary operation. *T-CONSTRAINT* indicates that the final type of the *constraints* expression is always the same as the original expression e_0. *T-HSKIP* and *T-VSKIP* indicate that the spatial parameter e is of the *len* type, and the final types of these two expressions should be *unit*. The last three of the typing rules are for the *union*, *struct* and *list* expressions, respectively.

$$\frac{\Gamma(x) = t}{\Gamma \vdash x : t} \quad \text{(T-VARIABLE)}$$

$$\frac{\Gamma \vdash e_1 : int \;\; \Gamma \vdash e_2 : int \;\; bop \in \{+, -, *, /, \%\}}{\Gamma \vdash e_1 \;\; bop \;\; e_2 : int} \quad \text{(T-INT ARITH)}$$

$$\frac{\Gamma \vdash e_1 : int \;\; \Gamma \vdash e_2 : int \;\; bop \in \{=, !=, <, >, <=, >=\}}{\Gamma \vdash e_1 \;\; bop \;\; e_2 : bool} \quad \text{(T-INT REL)}$$

$$\frac{\Gamma \vdash e_1 : float \;\; \Gamma \vdash e_2 : float \;\; bop \in \{+, -, *, /, \%\}}{\Gamma \vdash e_1 \;\; bop \;\; e_2 : float} \quad \text{(T-FLOAT ARITH)}$$

$$\frac{\Gamma \vdash e_1 : float \;\; \Gamma \vdash e_2 : float \;\; bop \in \{=, !=, <, >, <=, >=\}}{\Gamma \vdash e_1 \;\; bop \;\; e_2 : bool} \quad \text{(T-FLOAT REL)}$$

$$\frac{\Gamma \vdash e_0 : t_0, \Gamma \vdash e_1 : t_1, ..., \Gamma \vdash e_n : t_n}{\Gamma \vdash e_0(e_1, e_2, ..., e_n) : t_0} \quad \text{(T-CONSTRAINT)}$$

$$\frac{\Gamma \vdash e : len}{\Gamma \vdash hskip \;\; e : unit} \quad \text{(T-HSKIP)}$$

$$\frac{\Gamma \vdash e : len}{\Gamma \vdash vskip \;\; e : unit} \quad \text{(T-VSKIP)}$$

$$\frac{for \;\; each \;\; i \;\; \Gamma \vdash e_i : t_i}{\Gamma \vdash \{e_1|...|e_n\} : t_1 + ... + t_n} \quad \text{(T-UNION)}$$

$$\frac{for \;\; each \;\; i \;\; \Gamma \vdash e_i : t_i}{\Gamma \vdash \{e_1, ..., e_n\} : t_1 * ... * t_n} \quad \text{(T-STRUCT)}$$

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash e \;\; list : t \;\; list} \quad \text{(T-LIST)}$$

Figure 10: Inductive typing rules of ODL.

4. Robust ODL Parser

In this section, we focus on the ODL parser, which is the core of the entire information extraction system. We first describe the semantics that the parser is used for generating parsing trees based on the ODL specification. Afterwards, we describe the detail of the fuzzy matching strategy and the automatic correction model.

4.1. Semantics of the ODL Parser

Referring to Figure 8, the parsing process evaluates the entire data expression of ODL into hierarchical texts organized by the parsing tree, defined as $T = (node, [T_1, ..., T_n])$, where T_i is the i-th sub-tree of T. The leaf nodes of T are $(e, text)$ pairs representing the alignment between some primitive expression and its corresponding text, and non-leaf nodes are always the variable names x.

Since the entire expression has a complex structure, the evaluation is conducted recursively: it first evalu-

ates each sub-expression, and then composes multiple parsing trees into a large one. So the semantics of the parser consists of two parts: the evaluation rules for non-compositional expressions (constants, primitive variables and spatial expressions), and the inductive evaluation rules for compositional expressions.

The most basic rule is for evaluating constants (or primitive variables): given the OCR data D and the constant (or primitive variable) e in ODL, searching all possible alignments between e and some text box $d \in D$. During the whole parsing process, the parser maintains an environment $E = (x_0, y_0, x_1, y_1, x_{cr}, y_{cr})$, which contains the coordinates of the searching area, as well as a cursor (x_{cr}, y_{cr}). The cursor is a reference point, indicating the rough position that the desired text box resides. By default, the searching area is the whole image, and the cursor is at the top-left corner. The relative layout in ODL is represented in a left-to-right then top-to-bottom manner, thus the text box d becomes a candidate alignment of e, if it's within the searching area, and not located in the top-left side of the cursor. The following *InBound* function defines such rule:

$$\begin{aligned} InBound(D, E) = \{d \in D \mid & \\ & E.x_0 \le d.x_0 \le d.x_1 \le E.x_1 \\ \wedge & E.y_0 \le d.y_0 \le d.y_1 \le E.y_1 \\ \wedge & \neg(d.x_1 \le E.x_{cr} \wedge d.y_1 \le E.y_{cr})\}. \end{aligned} \tag{2}$$

With the environment E provided, the parser enumerates all candidate text boxes, and use a boolean match function $Match(e, d; E)$ to judge whether an alignment exists between d and the constant (or primitive variable) e. Intuitively, d is a valid match of e, if its text is close to the constraints of e, and it's located near the cursor $(E.x_{cr}, E.y_{cr})$. For a better flow of explanation, the formal definition of the match function will be given in the next section. If matches, a new parsing tree $T = ((e, d.text), [])$ is generated, which has a single node without any children. Besides, both D and E need to be changed, so that the parser can work on the alignment of the subsequent expressions in ODL. Following the relative layout of expressions, the cursor is moved to the top-right corner of the box: $E' = Move(E, d)$, and d is removed from D, as one text box can be aligned at most once: $D' = D - \{d\}$. Equation (3) defines a series of functions that changes the environment, which are used in different evaluation rules. In which, c is short for *coor*.

$$\begin{aligned} Move(E, d) &= (E.x_0, E.y_0, E.x_1, E.y_1, d.x_1, d.y_0), \\ Hskip(E, len) &= (E.x_0, E.y_0, E.x_1, E.y_1, E.x_{cr} + len, E.y_{cr}), \\ Vskip(E, len) &= (E.x_0, E.y_0, E.x_1, E.y_1, E.x_0, E.y_{cr} + len), \\ Restrict(c) &= (c.x_0, c.y_0, c.x_1, c.y_1, c.x_0, c.y_0). \end{aligned} \tag{3}$$

The tuple (T, E, D) is called a *parsing state*, which records the partial parsing tree, the environment information and the remaining text boxes. Since there exist multiple candidate text boxes for alignment, given E and D, e will be evaluated into a set of parsing states, written in the following judgment form:

$$E, D; e \Downarrow \bigcup_i \{(T_i, E_i', D_i')\}. \tag{4}$$

Based on the definition of the environment, match function and parsing state, Figure 11 lists all the evaluation rules. The rules E-Empty, E-PRIM1 and E-PRIM2 are used for evaluating the constants or primitive variables in a recursive form, where r stands for a list of parsing states. The parser generates new parsing trees based on whether d and e matches. In addition, the parser can simply skip the text box d and try to find alignments from remaining OCR data $D - \{d\}$.

The rules E-Hskip and E-Vskip are used for evaluating the spatial expressions. Intuitively, the *hskip* and *vskip* expression doesn't match any text boxes, but indicating the rough size of spacings. Therefore, both rules merely move the cursor horizontally or vertically, using $Hskip$ or $Vskip$ function defined in Equation (3).

Now we introduce the evaluation rules for compositional expressions, and focus on how the output parsing states are composed from the multiple sub-states. The rule E-Coor is used for evaluating spatial constraints $e(coor)$. The given coordinates *coor* restrict the searching area of alignments in the image, thus both the environment and the available OCR data are modified when evaluating e. The *Restrict* function sets the new environment based on *coor*, and the OCR data in the output parsing state contains all text boxes outside the searching area $(D - D')$, as well as unused boxes inside it (D_i''). There doesn't have a evaluation rule for value constraints, but such constraints will be used in the match function.

The last 5 rules in Figure 11 are used for evaluating *union*, *struct* and *list* expressions. The rules E-Wrap1 and E-Wrap2 construct the hierarchical structure of parsing trees, where the root node x is the identifier of union/struct/list expressions; E-Union simply combines parsing states from all its branches; E-Struct

$$\frac{InBound(D,E) = \emptyset}{E,D;e \Downarrow \{(((e,Nil),[]),E,D)\}} \quad \text{(E-Empty)}$$

$$\frac{d \in InBound(D,E), Match(e,d;E) = True, E' = Move(E,d)\ \ E', D-\{d\}; e \Downarrow r}{E,D;e \Downarrow r \cup \{(((e,d.text),[]),E',D-\{d\})\}} \quad \text{(E-PRIM1)}$$

$$\frac{d \in InBound(D,E), Match(e,d;E) = False, E' = Move(E,d)\ \ E', D-\{d\}; e \Downarrow r}{E,D;e \Downarrow r \cup \{(((e,Nil),[]),E',D-\{d\})\}} \quad \text{(E-PRIM2)}$$

$$\frac{E' = Hskip(E,len)}{E,D;hskip\ len \Downarrow \{(Nil,E',D)\}} \quad \text{(E-Hskip)}$$

$$\frac{E' = Vskip(E,len)}{E,D;vskip\ len \Downarrow \{(Nil,E',D)\}} \quad \text{(E-Vskip)}$$

$$\frac{E' = Restrict(coor), D' = InBound(D,E')\ \ E',D';e \Downarrow \bigcup_i\{(T_i,E_i'',D_i'')\}}{E,D;e(coor) \Downarrow \bigcup_i\{(T_i,(E.x_0,E.y_0,E.x_1,E.y_1,E_i''.x_{cr},E_i''.y_{cr}),D-D'+D_i'')\}} \quad \text{(E-Coor)}$$

$$\frac{E,D;e \Downarrow \bigcup_i\{(T_i,E_i',D_i')\}}{E,D;\{e\}\ as\ x \Downarrow \bigcup_i\{((x,[T_i]),E_i',D_i')\}} \quad \text{(E-Wrap1)}$$

$$\frac{E,D;e \Downarrow \bigcup_i\{(T_i,E_i',D_i')\}}{E,D;e\ list\ as\ x \Downarrow \bigcup_i\{((x,[T_i]),E_i',D_i')\}} \quad \text{(E-Wrap2)}$$

$$\frac{E,D;\{e_1\}\ as\ x \Downarrow r_1\ \ E,D;\{e_2|..|e_n\}\ as\ x \Downarrow r_2}{E,D;\{e_1|e_2|...|e_n\}\ as\ x \Downarrow r_1 \cup r_2} \quad \text{(E-Union)}$$

$$\frac{E,D;e_1 \Downarrow \bigcup_i\{(T_i,E_i',D_i')\}\ \ \forall i: E_i',D_i';\{e_2,...,e_n\}\ as\ x \Downarrow \bigcup_j\{((x,[T_{2ij},...,T_{nij}]),E_{ij}'',D_{ij}'')\}}{E,D;\{e_1,e_2,...e_n\}\ as\ x \Downarrow \bigcup_{ij}\{((x,[T_i,T_{2ij},...,T_{nij}]),E_{ij}'',D_{ij}'')\}} \quad \text{(E-Struct)}$$

$$\frac{E,D;e_1 \Downarrow \bigcup_i\{(T_i,E_i',D_i')\}\ \ \forall i: E_i',D_i';e\ list\ as\ x \Downarrow \bigcup_j\{((x,\boldsymbol{T}_{ij}^{(child)}),E_{ij}'',D_{ij}'')\}}{E,D;e\ list\ as\ x \Downarrow \bigcup_{ij}\{((x,[T_i]+\boldsymbol{T}_{ij}^{(child)}),E_{ij}'',D_{ij}'')\}} \quad \text{(E-List)}$$

Figure 11: Evaluation rules of the ODL parser.

binds the parsing tree of e_1 to the larger tree of the remaining parts; E-List is similar with E-Struct, considering that a list equals to a struct with unlimited expressions.

4.2. Fuzzy Match Function

The key feature of the parsing process is the robustness: rather than conducting exact match, the parser tolerates the OCR recognition errors, and the slight layout variances between images of the same format. In order to measure the degree of fuzzy matching between primitive expressions and text boxes, we define penalty functions at both value and spatial level, then based on that, we give the formal definition of the *Match* function.

Penalty Function for Value Constraints. The value penalty function $F_v(e, text)$ measures the penalty of aligning some text with the primitive expression e, i.e., either constants or primitive variables. For the constant c, since the desired value is fixed, the penalty score is simply derived from the edit distance (Levenshtein distance) between two texts, which measures the minimal number of edits required to change one text to the other. For the primitive variable x of the integer and floating point type, the value constraint is put into use. Referring to the value constraints $x(int, v_{min}, v_{max})$ and $x(float, length, precision, v_{min}, v_{max})$, edit distances are calculated between the text data and each numerical value, satisfying the restrictions of value range, length or precision, and the smallest edit distance is picked as the error score. The complete form of the value penalty $F_v(e, text)$ is defined as follows:

$$
\begin{aligned}
&F_v(c, text) = EditDist(c, text),\\
&F_v(x(int, v_{min}, v_{max}), text) = \min_i\{\\
&\quad EditDist(i, text) \mid i \in Z, v_{min} \le i \le v_{max}\},\\
&F_v(x(float, l, p, v_{min}, v_{max}), text) = \min_i\{\\
&\quad EditDist(i, text) \mid i \in R', v_{min} \le i \le v_{max}\},
\end{aligned} \tag{5}
$$

where R' is the set of all real numbers in length l and precision p. For example, the penalty score (edit distance) between the *constant* "Vent. rate" and the text "Vcnt. rule" is 3. As another example, the penalty score between $x(int, 60, 100)$ the text "53" is 1, since for all desired integers between 60 and 100, "63" holds the minimum edit distance with "53", which is 1.

Penalty Function for Spatial Layout. Recap that in the parsing process, based on the left-to-right and top-to-bottom relative layout between expressions embedded in ODL, the cursor $(E.x_{cr}, E.y_{cr})$ maintains a rough reference position that the desired text box resides. That's to say, the closer a text box d to the cursor, the higher confidence to align with the current expression. Therefore, the spatial penalty score $F_s(d, E)$ measures the spatial distance between the cursor and the top-left corner of box, calculated in L_1-norm:

$$
F_s(d, E) = |d.x_0 - E.x_{cr}| + |d.y_0 - E.y_{cr}|. \tag{6}
$$

Now we formally define the match function $Match(e, d; E)$. The function returns a boolean value for whether the text box d can be aligned to the primitive expression e. Given the above penalty functions at both value and spatial view, the *Match* function is defined as the weighted sum of two scores, controlled by an output threshold τ:

$$
Match(e, d; E) = \begin{cases} \text{T} & F_v(e, d.text) + k \cdot F_s(d, E) < \tau \\ \text{F} & \text{otherwise} \end{cases}. \tag{7}
$$

where k and τ are hyperparameters of the system. Finally, for picking the best parsing tree from multiple parsing states of the entire image expression, the textual error score of each primitive expression equally contributes to the final score of the whole parsing tree T. We define the overall score of T as follows:

$$
score(T) = \sum_{(e,text) \in leaf(T)} F_v(e, text). \tag{8}
$$

4.3. Automatic Correction Module

The correction module aims at automatically detecting and correcting error texts during the parsing process. For example, the parser not only detects the error of matching the text "15o" to the variable $p1$ in Figure 8, but also tries to correct the error text into "150" on-the-fly. We first explain the automatic correction in the parsing process, then discuss the incremental generation of the correction model.

4.3.1. Correction Model

The correction model M is a set of correction strategies S. Each correction strategy S is a probabilistic distribution of replacements from the original string *ori* to multiple candidate strings dst:

$$
S = \{(ori, dst_1, p_1), ..., (ori, dst_m, p_m)\}. \tag{9}
$$

A concrete correction strategy, for example, $S = \{(\text{"o"}, \text{"o"}, 0.6), (\text{"o"}, \text{"0"}, 0.3), (\text{"o"}, \text{"O"}, 0.1)\}$ indicates that given the character "o" there's a 60% possibility that no correction is needed, 30% possibility to replace into "0", and another 10% possibility to replace into "O". Since the most frequent error of OCR recognition happens at character level, all the original strings are short phrases (1,2,3-letter-gram). In addition, we define $rep(text, ori, dst)$ as the result of replacing all occurrences of *ori* with dst in the string *text*.

Given the correction model M, the parser is able to vary the input text such that a lower penalty for value constraints results. Intuitively, the value penalty between "15o" and $x(int)$ is always 1 without correction. It will become smaller than 1 when the correction model is applied, due to a possibility of varying "15o" to "150". More specifically, the relaxed version of value penalty F'_v is the minimum expectation score of different texts after being replaced by some strategy S:

$$
\begin{aligned}
F'_v(e, text; M) = \min_{S \in M}\{ &\sum_{(dst_i, p_i) \in S} p_i \cdot F_v(e, text'_i) \mid \\
&text'_i = rep(text, ori, dst_i)\}.
\end{aligned} \tag{10}
$$

By changing F_v in Equation (7) into F'_v, the matching function $Match(e, d; E, M)$ is able to make better judgments with the help of the correction model. Once some d and e matches (Figure 11, E-PRIM1), We record the best strategy $S^\star$ which reaches the minimum value penalty, and generate the corresponding parsing tree $T_i = ((e, d.text'_i), [])$ for each variant of string replacement by $S^\star$. For example, (p1, "15o"), (p1, "150") and (p1, "15O") are all valid parsing trees. The best parsing result will be picked by the overall scoring function defined in Equation (8). In this case, the system is able to

find the correct matching results from those candidate variants.

4.3.2. Generation of Correction Model

The model is generated in a hybrid approach. The initial correction model is generated from the result of OCR engine. For each text box, the OCR engine outputs top-K candidate texts, although only the best one is displayed in the raw OCR result. Regarding each i-th candidate as the replacement of the best text, the system counts all different (ori, dst) substring replacements. For example, given the replacement from "15o" to "150", distinct substring replacements are: ("1", "1"), ("5", "5"), ("o", "0"), ("15", "15"), ("5o", "50") and ("15o", "150"). For those images of the same format, the system counts all occurrences of substring replacements from every image, and builds the initial correction model M, where the probabilities are calculated by normalization over occurrences.

The correction model can be updated by incremental human corrections. Once the user manually changes some error texts into correct ones, the system obtains new substring replacements (ori, dst) and re-calculates all the probabilities in M. Since the human labeled texts are ground truth texts, all occurrences of substring replacements derived from human correction are multiplied by a weight factor w, so as to make larger contributions to the probability values in M.

5. Experiments

In this section, we introduce the medical image dataset and competing approaches for the structured information extraction task. We show the end-to-end extraction results, and investigate the effectiveness of automatic correction given manual annotations.

5.1. Experimental Setup

Medical Image Dataset. The medical image dataset which we use are from real ECG reports, which are recorded at different times and different hospitals. Those ECG reports can be divided into several different formats. Among them, we choose 4 typical image formats covering the most images, with examples shown in Figure 12. Though sharing similar medical information, the layout of textual data differ from each other. Table 1 lists the statistics of the dataset.

Table 1: Statistics of the ECG image dataset.

Format	1	2	3	4
Number of Images	124	113	102	97
Number of Attributes per Image	17	16	18	15

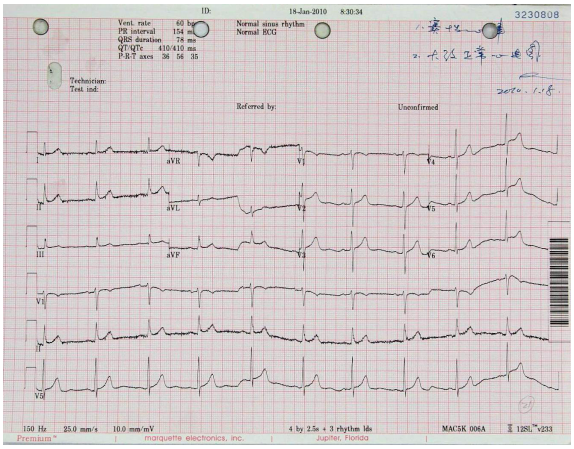
ODL Annotation. To obtain the data description of the medical images, we involve 6 volunteers to annotate the ODL of each image format. All the volunteers are undergraduate students in the major of computer science. The volunteer's training time is about 45 minutes, in which we present the syntax of ODL and several running examples to them. After the training process, each volunteer is asked to annotate the ODL of the 4 image formats in the dataset within 20 minutes. By manual inspection, 21 out of 24 ODL annotations are structural acceptable, where the defined data follows the left-to-right and top-to-bottom manner, and the spatial constraints are properly applied. Finally, we manually pick the best annotation of each image format, which are used in the experiments.

Image Preprocessing. The original medical images poses various problems which affects the performance of the OCR engine. For example, images are in different colors, and the textual information are surrounded by grid lines. To alleviate those problems, we binarize these images by a simple thresholding approach. We set separate thresholds for each of the RGB channels of the image, and combine the output values of all channels with AND operation. We make use of the auto-thresholding techniques in ImageJ Toolkit [8], which automatically determine the thresholds and generate the output images. Figure 13 shows the comparison between the original image and the corresponding one after preprocessing. By removing grid lines and turning to a binary image, the output is much cleaner, making the text on the image stand out.

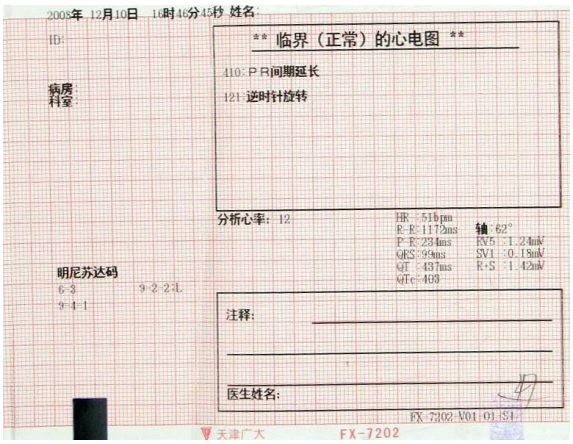
5.2. Evaluation of Structured Textual Information Extraction

Now we perform the end-to-end extraction task on the medical image dataset. We use the extraction accuracy over variables as the evaluation metric. For our proposed method (*ODL Parser*), we adopt Tesseract [2] as the OCR engine. We split the dataset into 5% for validation and 95% for testing. The hyperparameters k, τ and w are tuned by achieving the highest accuracy on the validation set.

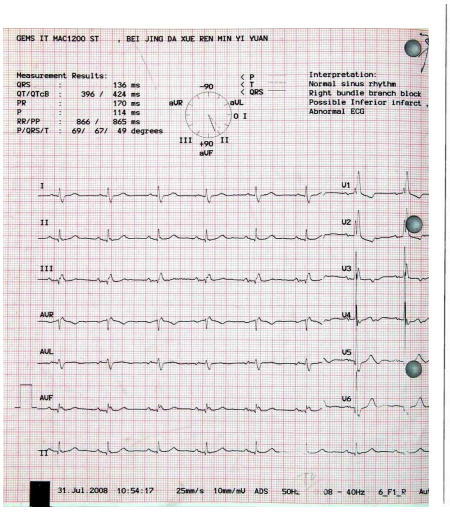
We compare our method with three competing methods discussed in Section 1. The first method (*Exact*



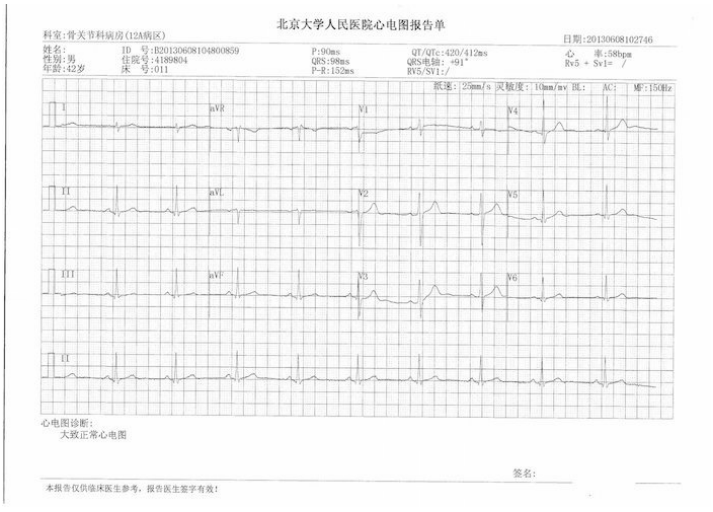
(a) Format 1



(b) Format 2



(c) Format 3



(d) Format 4

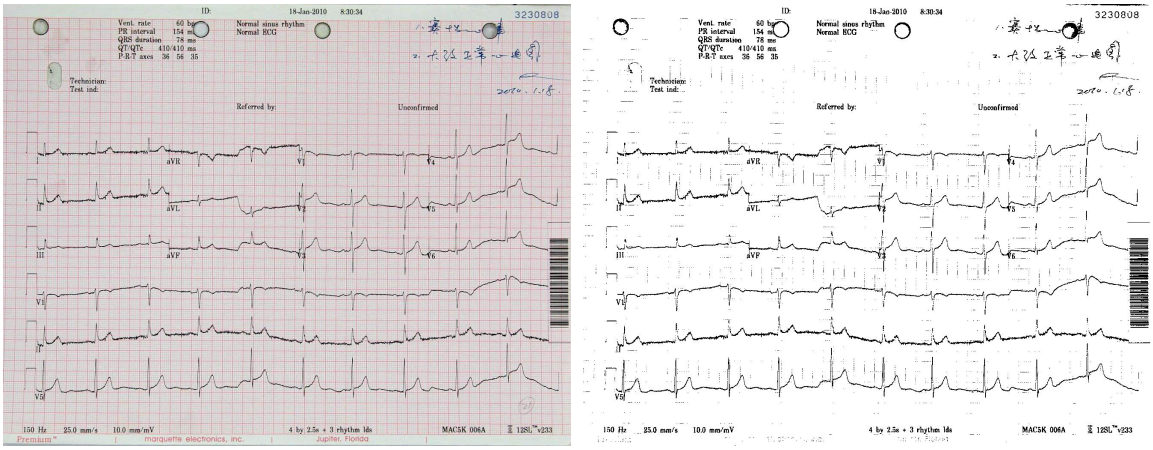
Figure 12: Example images of ECG formats in the dataset.

Match) is a textual matching approach. Based on the coordinates of each text box in the raw OCR result, we restore all text boxes into multiple rows of texts. Then we write simple regex rules to extract numerical variables by exactly matching the constant strings before and after them. No fuzzy matching or data constraints are applied in this method.

The second competing method (*Zonal OCR*) is based on manually marking zones of each data on the images. Tesseract is applied for recognizing texts from each image fragment. In order to overcome the slight positional variations between images, the annotating process is improved in this method. As shown in Figure 14, for images of the same format, we first label all zones of interest in one image (the blue boxes), as well as another referent zone (the red box). We keep the relative distance between all boxes, and only label the referent zone for all the remaining images. Therefore, all zone of interests of the remaining images can be automatically calculated.

The third approach (*Page Layout*) is based on the page layout analysis, with examples depicted in Figure 4. We adopt ABBYY recognizer for page layout analysis, and implement ad-hoc rules to map the desired data with specific text units in the page layout result.

The experiments are conducted on the machine with Intel i7-4790 CPU and 32G memories. The ODL parser parsers each image in 2 seconds on average, and the memory consumption doesn't exceed 1G. The performance can be further boosted by multiple threading.



(a) Before preprocessing (b) After preprocessing

Figure 13: Example of image preprocessing.

Table 2 shows the experimental results over the testing images. Our ODL-based approach consistently outperforms existing approaches by an absolute gain of 3-6% on all the 4 ECG formats, which demonstrates the effectiveness of the robust parsing process.

5.3. Evaluation of Automatic Correction

In this section, we conduct the additional experiments to analyze the performance gain brought by human correction in our system. For each image format, the system prompts a certain number of imperfect $(e, d.text)$ alignments as parsing errors, collects all manual correction data, and performs a new run of parsing based on the updated correction model. Given the set of all detected parsing errors, we compare two policies of recommending errors for manual correction: *random* and *most frequent*. For the *random* policy, the system follows the 2-step sampling without replacement: first randomly samples e from all distinct expressions with parsing errors, then randomly selects one instance from all parsing errors related to e. The *most frequent* policy is also based on sampling without replacement, while for each pick, the system always randomly samples the parsing error from those expressions with the most parsing errors.

Table 2: Accuracy of structured information extraction over medical images.

Format	1	2	3	4
Exact Match	58.8%	56.3%	61.1%	53.4%
Zonal OCR	81.2%	79.8%	81.7%	80.6%
Page Layout	79.7%	80.2%	81.2%	81.1%
ODL Parser	**85.5%**	**83.8%**	**84.9%**	**84.0%**

Figure 15 reports the extraction accuracies with 0, 1, 5, 10, 15 human corrections using different recommending policies. For each specification, we conduct the experiment 100 times, and report the average accuracy. Due to the intensive labour in repeating experiments, we didn't perform experiments on larger human corrections. For all different image formats, the more corrections we make, the better accuracy we can get, which demonstrates the effectiveness of the correction model. The improvement to the extraction accuracy is better when using the *most frequent* policy instead of the *random* baseline. We argue that by paying more attentions to the most common recognitions errors, the correction model under the *most frequent* recommendation is able to correct more imperfect alignments, rather than wasting limited corrections to long-tail cases. In addition, for the accuracy curve of the *most frequent* policy, we observe that the growing rate goes down when increasing the number of human corrections. This is reasonable, as compared with frequent recognitions errors, the incremental corrections on infrequent ones lead to fewer improvements.

5.4. Discussion and Analysis

This section compares and contrasts the ODL based model with other baselines approaches in the end-to-end extraction task. Based on the results shown in Table 2, we first observe that *Exact Match* baseline is outperformed by the other approaches by a significant margin. The result is not surprising, due to that exact matching

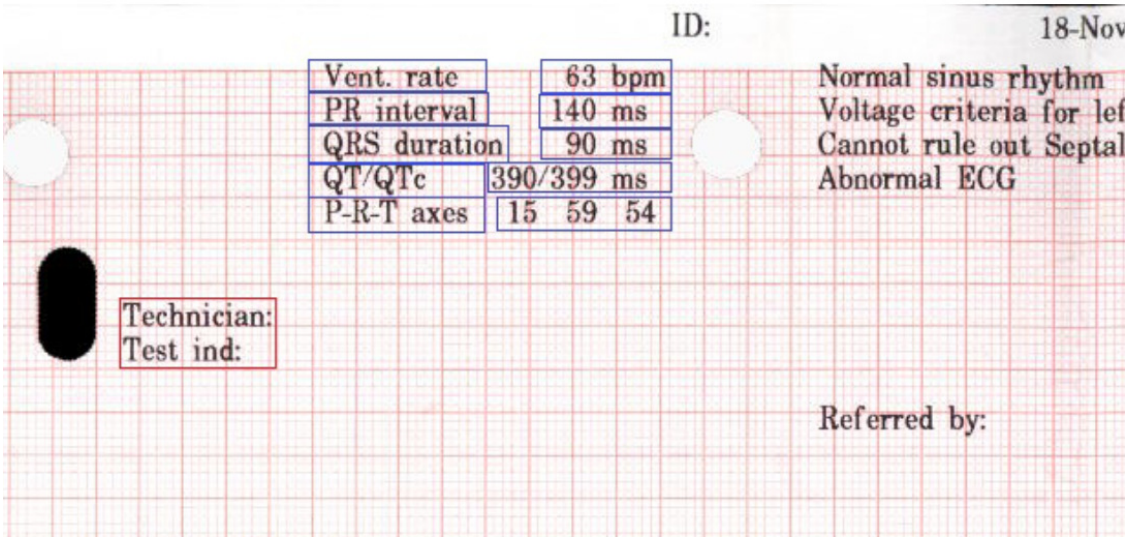


Figure 14: Example of ECG image with zones of interests and referent zones.

approach is highly sensitive to the recognition accuracy. Errors occurring on either the target variable or its surrounding constants will lead to a matching failure, and noisy text boxes become another factor to affect the extraction accuracy.

The other two baselines, though marginally outperformed by the ODL-based approach, still have their own important limitations. *Zonal OCR* is a typical approach that greatly simplifies the parsing process, at the expense of intensive human labour. As shown in Figure 14, users have to annotate the referent zone (the red box) for *every* single image, which is inefficient. Besides, the main assumption of this approach is that relative layouts of texts are accurate and unchanged among all the images of the same format. The assumption makes sense, but doesn't always hold. Misrecognition of the zone is disastrous, as all the extracted information will be incorrect.

For the *Page Layout* approach, it also requires the consistent page layout result for images of the same format. Figure 4 gives an example of correct page layout, however, if the textual zone are recognized incorrectly, necessary information may be omitted from output. As shown in Figure 16, the largest textual zone is imperfectly located, which leads to error information extraction results. Although having a relative high accuracy, the extraction wrapper of this method is rather ad-hoc, which is more challenging for putting into real use.

Compared with those baselines, the ODL based approach doesn't rely on any carefully defined bounding boxes, its most important advantage is to make full use of the relative layout information between expressions embedded in the ODL description. Based on the implicit layout restrictions, the parser is able to generate the optimized alignment between expressions and text boxes, while satisfying both value and spatial constraints.

6. Related Work

6.1. Textual Information Extraction

There has been a lot of research works for extracting textual information from various images, such as book covers, digital drawings, natural scenes, and video frames [1]. More recent researches focus on the text recognition in specific domains, such as the character detection of natural scenes [9], which can be further improved by leveraging the symmetric property of characters [10]. The ICDAR2017 Robust Reading Challenge [11] aims at extracting texts from biomedical literature figures, including the task of text detection and cropped word recognition.

With the growth of text recognition methods, several researches belong to the downstream applications using OCR techniques, such as content based image retrieval [12], and lossless image compression [13]. Our presented work is regarded as another downstream application, which converts the unstructured recognized texts into the structured knowledge of medical records. The OCR engine serves as the building block of the system, but the main contribution of this work goes beyond the scope of OCR.

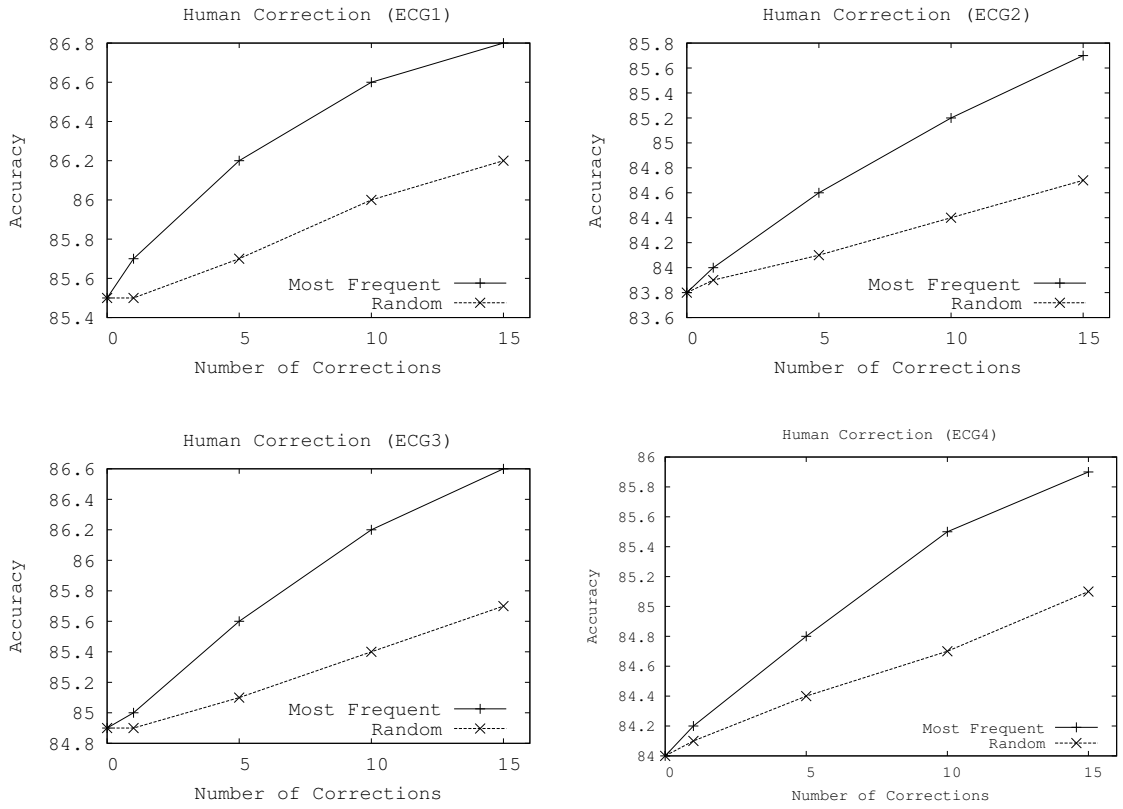


Figure 15: Extraction accuracy with human annotated correction.

Another related research field is information extraction from free texts. A series of OpenIE researches [14, 15] extract the (subject, predicate, object) relation triples from the natural language sentences in the open domain. There also exists information extraction system in specific domains, such as cTAKES [16] in the clinical domain and BioInfer [17] in the biomedical domain. While the goal of these tasks are learning extraction rules for textual knowledge, our work targets on a different perspective: The structured knowledge depends on the user's demand, rather than concrete facts or pre-defined relations. Besides, the input texts of our work are separated text boxes with recognition errors, which is far more challenging than ordinary free texts. Therefore, our research targets on using description languages for data representation and parsing, which is a non-trivial task.

6.2. Data Description Language

In previous work, some declarative data description languages were designed for different purposes. For example, PADS [7] is a data description language to handle the ad-hoc log data. With the help of the descriptions, a compiler can be used to parse and print the data. Further research including the LearnPADS [18, 19], which provides a fully automatic system for generating the corresponding PADS descriptions. ODL is inspired by PADS and uses the type system in programming language integrated with fuzzy matching and spatial features to handle the specific text data from medical images.

In ODL, in order to describe spatial information, we enhance the syntax with spatial features in order to limit the search area of the description and horizontal and vertical skip to describe the spatial relation between the data. There has also been some previous work on describing the spatial information in the document, such as LaTeX [20] and PostScript [21]. In LaTeX, lots of spacing parameters and spacing commands are used. For example, "\vspace{⟨skip⟩}" and "\hspace{⟨skip⟩}" are two general spacing commands. In PostScript, in order to manipulate the text, some operations are designed, including *ashow*, *widthshow*, *awidthshow* and so on. These operations take an input text string and a

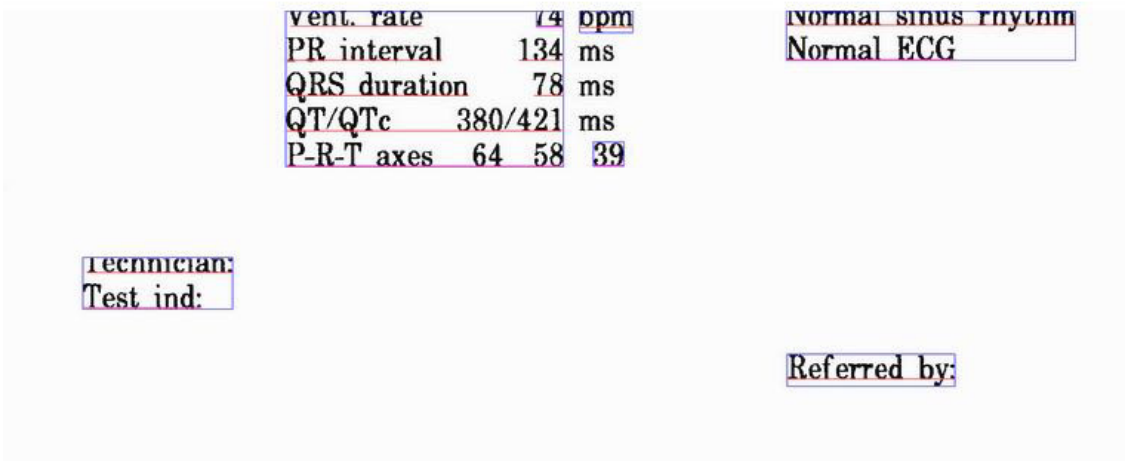


Figure 16: Example result of an imperfect page layout analysis.

separate specification for positioning the elements. We use similar descriptions for spacing functions, "*hskip len*" and "*vskip len*". But the way we interpret them is different. The reason is ODL is a fuzzy description language so that it's easy for humans to write in ODL. ODL will tolerate some error of the spacing command by using the fuzzy matching strategies. However, in LaTeX, spacing commands are interpreted as it is.

6.3. Page Layout Analysis

A typical document analysis system consists of page segmentation, optical character recognition, and logical structure identification. The interest in the logical structure was inspired by the emergence and popularity of common representation standards such as XML. By using these common standards, we can encode structural information together with the contents. [6]. One of the key components used to help to understand a document is logical labeling. The task of logical labeling is to label segmented blocks on a document image as title, author, header, text column, etc [22]. The set of labels will depend on the document classes or applications. Logical labeling techniques can be roughly characterized as either zone-based or structure-based. Zone-based techniques classify zones individually based on the features of each zone [23] [24]. Structure-based techniques incorporate global constraints such as the position of the text. In our work, instead of using zone or struct, we describe the logical layout by making use of the composition expressions.

Some style-directed layout analysis algorithms also allow users to specify the physical layout [25]. In this case, a regular language, including terminal symbols, non-terminal symbols and production rules, is proposed to express the varieties of physical regions and help the physical layout analysis. However, the basic terminal symbol is the text line instead of the word. The reason is that it is hard to distinguish word from noises based on the image features alone, which result in inaccuracies when processing the production rules. In our work, we introduce constraints to handle such noises.

Another related work about page segmentation is VIPS [26]. It's a vision-based page segmentation algorithm used to extract the semantic structure of a web page. In our work, the semi-structure output of the OCR engine is comprised of the structured XML files. However, VIPS can't be directly used to analyze these XML files since the errors in the recognition process will lead to errors in the XML files about the visual cues and the DOM structures.

Different from the solutions for web pages, PATO [27] is a system for extracting predefined items from printed documents in a dynamic, multisource scenario. By focusing on the parameters of the text blocks, it pays little attention to the relationship between different text blocks, that is, it is not able to represent dependencies between text block, while our system does.

7. Conclusion

In this paper, we have proposed ODL, the declarative data description language for describing and extracting structured textual information from various medical images. The syntax of ODL makes use of both value and layout information to describe the data format of images, and the proposed ODL parser then gen-

erates the best alignments between structured data in ODL and the texts recognized by OCR engine. As the key feature of our system, the parsing phase is built upon the fuzzy matching between ODL expressions and text boxes, and such tolerance of imperfect alignment leads to the more robust information extraction process. In addition, based on multiple candidate texts of the OCR engine and manually annotated corrections, the correction model is combined with ODL parser, which is able to correct frequent recognition errors on-the-fly. We collected the ECG image dataset for evaluation, and the end-to-end experimental results demonstrate that our ODL-based approach consistently outperforms the other existing solutions. Furthermore, the evaluation of the manual correction correction indicates that by using the frequency based policy to prompt parsing errors, the extraction accuracy can be effectively improved with limited manual corrections. Finally, though the data used in this paper are exclusively medical images, which is the preliminary study of our research, this framework will conceivably benefit structured information extraction tasks targeting the other types of semi-structured images.

Acknowledgments

Kenny Q. Zhu and Jia Wei are the corresponding authors. This work was partially supported by the AstraZeneca-SJTU collaborative research grant.

References

[1] K. Jung, K. I. Kim, A. K. Jain, Text information extraction in images and video: a survey, Pattern Recognition 37 (5) (2004) 977–997 (2004).

[2] R. Smith, An overview of the tesseract OCR engine., in: Proceedings of the International Conference on Document Analysis and Recognition, Vol. 7, 2007, pp. 629–633 (2007).

[3] R. Smith, D. Antonova, D.-S. Lee, Adapting the tesseract open source OCR engine for multilingual OCR, in: Proceedings of the International Workshop on Multilingual OCR, ACM, 2009, pp. 1–8 (2009).

[4] L. Vincent, Google book search: Document understanding on a massive scale, in: Proceedings of the International Conference on Document Analysis and Recognition, Vol. 2, IEEE Computer Society, 2007, pp. 819–823 (2007).

[5] M. Lebert, Project gutenberg (1971-2008) (2008).

[6] L. O'Gorman, The document spectrum for page layout analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (11) (1993) 1162–1173 (1993).

[7] K. Fisher, R. Gruber, PADS: a domain-specific language for processing ad hoc data, ACM Sigplan Notices 40 (6) (2005) 295–304 (2005).

[8] C. A. Schneider, W. S. Rasband, K. W. Eliceiri, NIH Image to ImageJ: 25 years of image analysis, Nature Methods 9 (7) (2012) 671 (2012).

[9] X.-C. Yin, X. Yin, K. Huang, H.-W. Hao, Robust text detection in natural scene images, IEEE Transactions on Pattern Analysis and Machine Intelligence 36 (5) (2014) 970–983 (2014).

[10] Z. Zhang, W. Shen, C. Yao, X. Bai, Symmetry-based text line detection in natural scenes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2558–2567 (2015).

[11] C. Yang, X.-C. Yin, H. Yu, D. Karatzas, Y. Cao, ICDAR2017 robust reading challenge on text extraction from biomedical literature figures (DeTEXT), in: Proceedings of the International Conference on Document Analysis and Recognition, Vol. 1, IEEE, 2017, pp. 1444–1447 (2017).

[12] P. A. Wankhede, S. W. Mohod, A different image content-based retrievals using OCR techniques, in: Proceedings of the International Conference of Electronics, Communication and Aerospace Technology, Vol. 2, IEEE, 2017, pp. 155–161 (2017).

[13] A. Ergüzen, E. Erdal, Medical image archiving system implementation with lossless region of interest and optical character recognition, Journal of Medical Imaging and Health Informatics 7 (6) (2017) 1246–1252 (2017).

[14] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, T. M. Mitchell, Toward an architecture for never-ending language learning., in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 5, Atlanta, 2010, p. 3 (2010).

[15] A. Fader, S. Soderland, O. Etzioni, Identifying relations for open information extraction, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2011, pp. 1535–1545 (2011).

[16] G. K. Savova, J. J. Masanz, P. V. Ogren, J. Zheng, S. Sohn, K. C. Kipper-Schuler, C. G. Chute, Mayo clinical text analysis and knowledge extraction system (cTAKES): architecture, component evaluation and applications, Journal of the American Medical Informatics Association 17 (5) (2010) 507–513 (2010).

[17] S. Pyysalo, F. Ginter, J. Heimonen, J. Björne, J. Boberg, J. Järvinen, T. Salakoski, Bioinfer: a corpus for information extraction in the biomedical domain, BMC bioinformatics 8 (1) (2007) 50 (2007).

[18] K. Fisher, D. Walker, K. Q. Zhu, P. White, From dirt to shovels: fully automatic tool generation from ad hoc data, ACM SIGPLAN Notices 43 (1) (2008) 421–434 (2008).

[19] K. Fisher, D. Walker, K. Q. Zhu, LearnPADS: automatic tool generation from ad hoc data, in: Proceedings of the SIGMOD International Conference on Management of Data, ACM, 2008, pp. 1299–1302 (2008).

[20] L. Lamport, A. LaTEX, Document Preparation System, Addison-Wesley Reading, MA, 1986 (1986).

[21] E. Taft, S. Chernicoff, C. Rose, Post-Script Language Reference, Addison-Wesley, 1999 (1999).

[22] J. Liang, D. Doermann, Logical labeling of document images using layout graph matching with adaptive learning, in: Proceedings of the International Workshop on Document Analysis Systems, Springer, 2002, pp. 224–235 (2002).

[23] O. Altamura, F. Esposito, D. Malerba, Transforming paper documents into XML format with WISDOM++, International Journal on Document Analysis and Recognition 4 (1) (2001) 2–17 (2001).

[24] G. I. S. Palmero, Y. A. Dimitriadis, Structured document labeling and rule extraction using a new recurrent fuzzy-neural system, in: Proceedings of the International Conference on Document Analysis and Recognition, IEEE, 1999, pp. 181–184 (1999).

[25] T. Kanungo, S. Mao, Stochastic language models for style-directed layout analysis of document images, IEEE Transactions on Image Processing 12 (5) (2003) 583–596 (2003).

[26] D. Cai, S. Yu, J.-R. Wen, W.-Y. Ma, VIPS: A vision-based page segmentation algorithm, Tech. rep., Microsoft technical report, MSR-TR-2003-79 (2003).

[27] A. Bartoli, G. Davanzo, E. Medvet, E. Sorio, Semisupervised wrapper choice and generation for print-oriented documents, IEEE Transactions on Knowledge and Data Engineering 99 (2014) 1–14 (2014).