PICK: Processing Key Information Extraction from Documents using Improved Graph Learning-Convolutional Networks

Wenwen Yu¹, Ning Lu², Xianbiao Qi², Ping Gong¹, Rong Xiao²

¹Xuzhou Medical University, Xuzhou, China
²Ping An Property & Casualty Insurance Company, Shenzhen, China

Task Definition

• Key Information Extraction (KIE) from documents is the downstream task of OCR.
• The aim of KIE is to extract a number of key fields from the given documents, and save the texts to structured documents.
• KIE is essential for a wide range of technologies such as efficient archiving, fast indexing, document analysis and so on.

Motivation

• KIE is a challenge task because documents not only have textual features extracting from OCR systems, but also have semantic visual features that are not fully exploited, and it play a critical role in KIE.
• Too little work has been devoted to efficiently make full use of both textual and visual features of the documents.
• Existing methods for KIE only use text and box, and need task-specific knowledge and human-designed rules.

Method

The overall architecture is shown in Figure 2, which contains 3 modules:

• Encoder: This module encodes textual and morphology information individually, which will be used as node input to the Graph Module.
• Graph Module: This module can catch the latent relation between nodes and gather richer graph embeddings representation of nodes through improved graph learning-convolutional operation, which get non-local and non-sequential features.
• Decoder: This module performs sequence tagging on the union non-local sentence at character-level using BiLSTM and CRF, respectively.

Graph Learning

Given an input \( \mathbf{V} = [v_1, \ldots, v_N] \in \mathbb{R}^{N \times d_{\text{token}}} \) of graph nodes, where \( v_i \in \mathbb{R}^{d_{\text{token}}} \) is the \( i \)-th node of the graph, Graph Module generate a soft adjacent matrix \( \mathbf{A} \) that represents the pairwise relationship weight between two nodes.

\[
A_{ij} = \text{softmax}(\mathbf{w}_j^T v_i), \quad i = 1, \ldots, N, \quad j = 1, \ldots, N.
\]

where \( \mathbf{w}_j \in \mathbb{R}^{d_{\text{token}}} \) is learnable weight vector.

Graph Convolution

Firstly, given an input \( \mathbf{V}_0 = [v_0, \ldots, v_N] \in \mathbb{R}^{N \times d_{\text{token}}} \) as the initial layer input of the graph, initial relation embedding \( \mathbf{a}_{ij}^0 \) between the node \( v_i \) and \( v_j \) is formulated as follows:

\[
\mathbf{a}_{ij}^0 = \mathbf{W}_0^T [v_i, v_j, \delta(v_i, v_j), \delta(v_i, v_j)]\quad (4)
\]

where \( \mathbf{W}_0 \in \mathbb{R}^{d_{\text{model}} \times 6} \) is learnable weight matrix. Then we extract hidden features \( h_i^l \) between the node \( v_i \) and \( v_j \) from the graph using the node-edge-node triplets \( (v_i, \alpha_{ij}, v_j) \) data in the \( l \)-th convolution layer, which is computed by

\[
R_{ij}^l = \sigma(h_i^l \mathbf{w}_j^T + h_j^l \mathbf{w}_i^T + \mathbf{a}_{ij}^l + h_i^j)\quad (5)
\]

Finally, node embedding \( \mathbf{h}_i^{l+1} \) aggregate information from hidden features \( h_i^l \) using graph convolution to update node representation. For node \( v_i \), we have

\[
\mathbf{h}_i^{l+1} = \sigma(\mathbf{W}_l \mathbf{h}_i^l),\quad (6)
\]

where \( \mathbf{W}_l \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}} \) is layer-specific learnable weight matrix in the \( l \)-th convolution layer.

The relation embedding \( \mathbf{a}_{ij}^{l+1} \) in the \( l+1 \)-th convolution layer for node \( v_i \) is formulated as

\[
\mathbf{a}_{ij}^{l+1} = \sigma(\mathbf{W}_l \mathbf{a}_{ij}^l),\quad (7)
\]

where \( \mathbf{W}_l \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}} \) is layer-specific trainable weight matrix in the \( l \)-th convolution layer.

Results

Table 1: Performance comparison between PICK (Ours) and baseline method on Medical invoice datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Baseline mEP</th>
<th>PICK (Ours) mEP</th>
<th>mER</th>
<th>mER (Ours) mEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT</td>
<td>66.8</td>
<td>92.1</td>
<td>85.0</td>
<td>92.1</td>
</tr>
<tr>
<td>CCTA</td>
<td>85.7</td>
<td>92.4</td>
<td>92.4</td>
<td>92.4</td>
</tr>
<tr>
<td>IN</td>
<td>61.1</td>
<td>92.4</td>
<td>92.4</td>
<td>92.4</td>
</tr>
<tr>
<td>Name</td>
<td>53.4</td>
<td>68.7</td>
<td>68.7</td>
<td>78.9</td>
</tr>
<tr>
<td>HN</td>
<td>69.3</td>
<td>92.4</td>
<td>92.4</td>
<td>92.4</td>
</tr>
<tr>
<td>Overall (micro)</td>
<td>71.4</td>
<td>78.1</td>
<td>85.0</td>
<td>87.0</td>
</tr>
</tbody>
</table>

Table 2: Results of each component of our model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Medical Invoice (mEP)</th>
<th>Train Ticket (mEP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICK (Full model)</td>
<td>87.8</td>
<td>92.4</td>
</tr>
<tr>
<td>w/o image segments</td>
<td>[0.9]</td>
<td>[0.0]</td>
</tr>
<tr>
<td>w/o graph learning</td>
<td>[1.0]</td>
<td>[0.7]</td>
</tr>
</tbody>
</table>

https://github.com/wenwenyu/PICK-pytorch