Economics 2355: Labeling

Melissa Dell

Harvard University

February 2021

Outline

Introduction

Active Learning for Layout Annotation

Labeling Hacks

Introduction

- A major challenge of deep learning is that it is data hungry. If you want your model to learn your document layouts, you have to give it the information to do so.
- Sometimes there are ways to get around labeling altogether, by using self-supervised learning.
- In the case of recognizing complex document layouts, however, right now there are not methods that allow you to get around needing to label.

Self-Supervised Potential

- This would possibly look something like using a model, i.e. CycleGan, to turn auto-generated documents (created with a word processor) into documents that look like the actual documents you want to process
- A DL model would learn to add noises to the clean simulated documents that mimic the real world noises in the actual documents
- We are not aware of any work attempting to implement this kind of approach, but we do think we will see a lot more along these lines in coming years

Outline

Introduction

Active Learning for Layout Annotation

Labeling Hacks

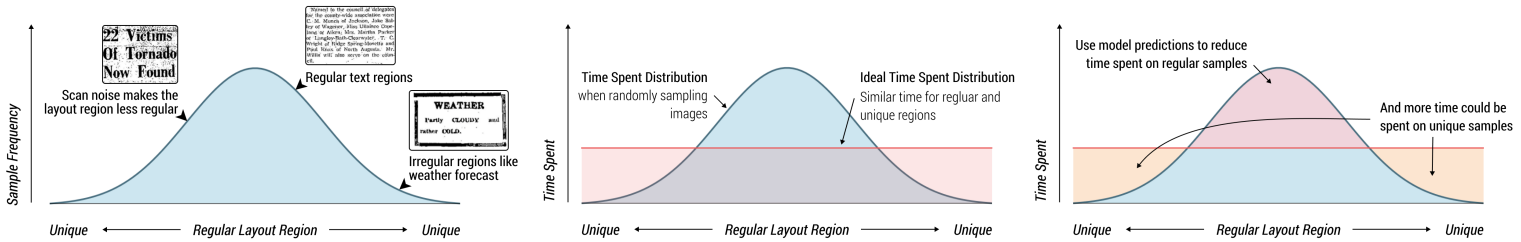
Overview

- Document images often have intricate layout structures, with numerous content regions (texts, figures, tables) densely arranged on each page.
- This makes the manual annotation of layout datasets expensive and inefficient.

Active Learning

- Active learning prioritizes the most important samples to annotate
- AL methods typically score and select samples at the **image** level, rather than at the **object** level
- For category-imbalanced layout images, this could suffer from the over-exposure of common objects, leading to suboptimal results
- Other methods are inapplicable to documents, i.e. because they assume more space without objects than is typically found in dense document images

Challenge

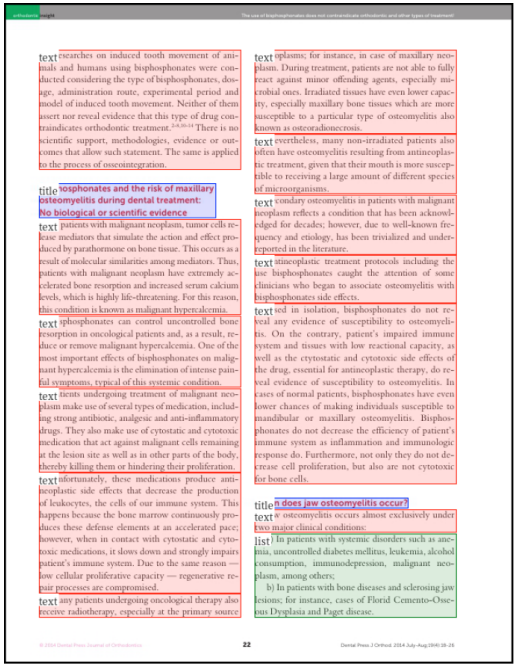


1. Layout Region Distribution | 2. Time Spent During Annotation | 3. The Function of Model Predictions

Unpublished figure created by the author.

- Layout regions can follow a long tailed distribution, where the frequency of different samples differs substantially.
- If we randomly sample images for labeling, the time distribution is approximately the same as the layout region distribution
- We'd like to oversample the tails, but the layout distribution is unobserved ex ante.

Example



PubLayNet Document Page

HJDataset Document Page

PRImA Document Page

Figure: Three exemplar document layouts from PublayNet, HJDataset, and PRImA. There are numerous layout objects per page, many of them are very similar and from the same category. Directly labeling them all will result in waste of precious labeling budget (Shen et all, 2020).

Our Solution

- Inspired by recent progresses in semi-supervised learning and self-training, we (Shen, Zhao, Dell, Yu, and Li) propose an **O**bject-**L**evel **A**ctive **L**earning framework for efficient document layout **A**nnotation, `OLALA`, the first (to our knowledge) AL framework designed specifically for documents.
- In this framework, only regions with the most ambiguous object predictions within an image are selected for annotators to label, optimizing the use of the annotation budget.
- For unselected predictions, a semi-automatic correction algorithm is proposed to identify certain errors based on prior knowledge of layout structures and to rectify them with minor supervision.

Method

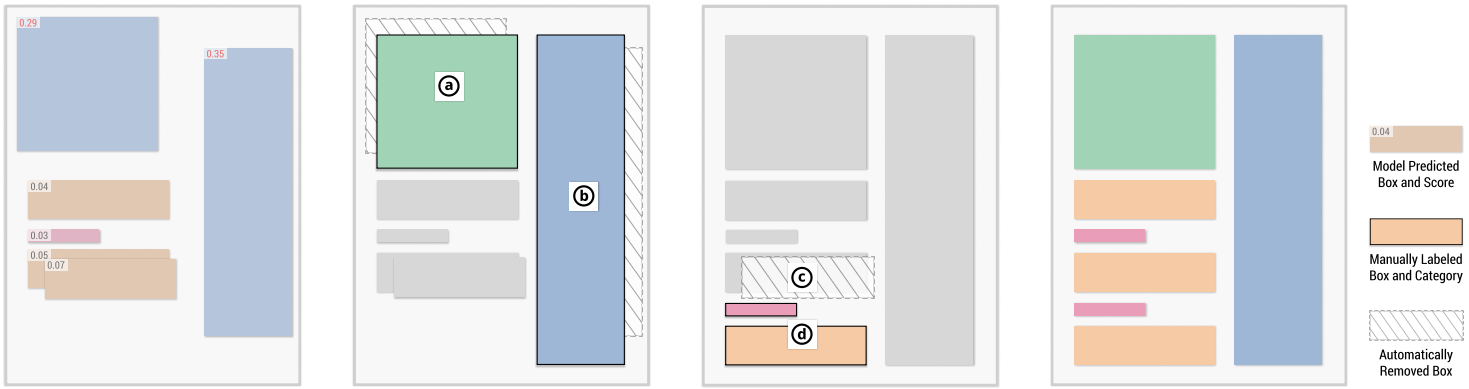


Figure: Illustration of the OLALA framework (Shen et all, 2020). [1] During labeling, for an input image, a trained model predicts the layout with various errors. A object scoring function *f* evaluates the informativeness for each object prediction. [2] OLALA selects the regions of top scores and sends them for manual labeling to correct wrong object category (a) and bounding box (b). [3] A semi-automatic prediction correction algorithm is applied to rectify duplicated object (c) and recover false-negatives (d) with minimal extra supervision. [4] After this process, the final annotation is obtained with labeling only a part of the objects.

Advantages of Active Learning

- For a given labeling budget, substantially higher accuracy can be achieved with active learning
- A dataset of the same size can be created with significantly less human effort

Object-Level Scoring

- At the core of object level active learning is an object scoring function.
- We design a perturbation-based object scoring function for document images.
- It governs the object selection process via evaluating prediction ambiguities.
- It considers both the positions and categories of predicted layout objects.

Object-Level Scoring

- The scoring function evaluates prediction ambiguity and determines which objects to select for labeling
- We use a perturbation-based scoring method based on both the bounding box and category predictions to account for specific characteristics in layout detection tasks
- The method hypothesizes that the adjacent image patches share similar features vectors, and the predicted object boxes and categories for them should be consistent
- Any large disagreement between the original and perturbed predictions indicates that the model is insufficiently trained for this type of input, or there is some anomalies in the given sample, both of which demand user attention

Perturbation-based Scoring Function

- Specifically, for each object prediction $\hat{y}_j = (\hat{b}_j, \hat{c}_j) \in \hat{Y}_i$, we take the bounding box prediction $\hat{b}_j = (x, y, w, h)$ and apply some small shifts to perturb the given box
- The new boxes are created via horizontal and vertical translation by a ratio of α and β: $p_{jk} = (x \pm \alpha w, y \pm \beta h, w, h)$, where p_{jk} is the k-th perturbed box for box prediction $\hat{b}_j$, and a total of K perturbations will be generated.
- Based on features within p_{jk}, the model generates new box and category predictions (q_{jk}, v_{jk}).
- We then measure the disagreement between the original prediction $(\hat{b}_j, \hat{c}_j)$ and the perturbed versions $\{(q_{jk}, v_{jk})\}_{k=1}^{K}$, and use it as a criterion for selecting objects for labeling.

Perturbation-based Scoring Function

- In practice, we build this method upon a typical object detection architecture composed of a region proposal network and a region classification and improvement network, ROIHeads, that predicts the category and modifies the box prediction based on the input proposals.
- We use the perturbed boxes $\{p_{jk}\}_{k=1}^{K}$ as the new inputs for the ROIHeads, and obtain the new box and class predictions $\{(q_{jk}, v_{jk})\}_{k=1}^{K}$.

Perturbation-based Scoring Function

We formulate the position disagreement D_p and the category disagreement D_c for the j-th object prediction as

$$D_p(\hat{b}_j) = \frac{1}{K} \sum_k \Big(1 - \text{IOU}(\hat{b}_j, p_{jk})\Big)$$

$$D_c(\hat{c}_j) = \frac{1}{K} \sum_k L(\hat{c}_j || v_{jk}),$$

where IOU calculates the intersection over union scores for the inputs, and $L(\cdot||\cdot)$ is a measurement for distribution difference, e.g., cross entropy.

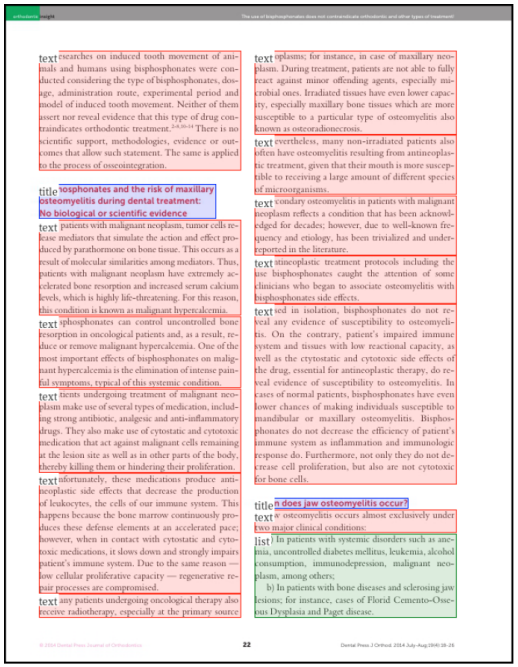
The overall disagreement is $D(\hat{y}_j) = D_p(\hat{b}_j) + \lambda D_c(\hat{c}_j)$, with λ being a weighting constant.

Objects of larger D will be prioritized for labeling

Perturbation-based Scoring Function

- The proposed method evaluates the box and category prediction robustness, and can effectively identify false-positive object predictions.
- Incorrect category prediction $\hat{c}_j$ will tend to cause high D_c due to divergence of the new class prediction v_{jk} for nearby patches.
- When the predicted box $\hat{b}_j$ is wrong, the perturbed box p_{jk} is less likely to be the appropriate proposal.
- The generated predictions (q_{jk}, v_{jk}) are unreliable, causing higher overall disagreement D.

Testing the Approach



PubLayNet Document Page

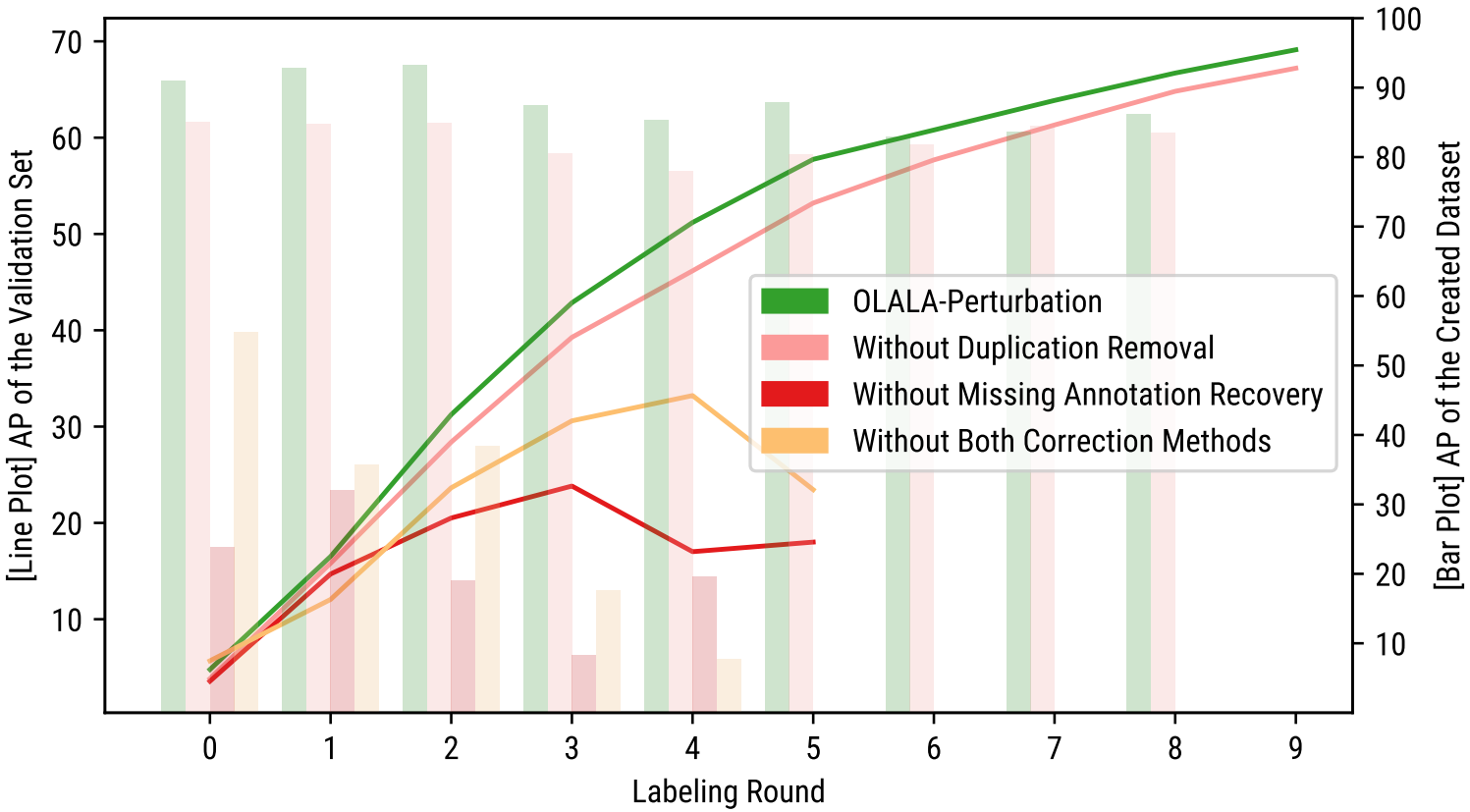
HJDataset Document Page

PRImA Document Page

Figure: We run labeling simulations on three datasets: PubLayNet (360K images), PRImA (453 images), and HJDataset (2K images) (Shen et al, 2020).

Results Summary

Paper has various tables and figures, comparing to other methods, but this is the most important:



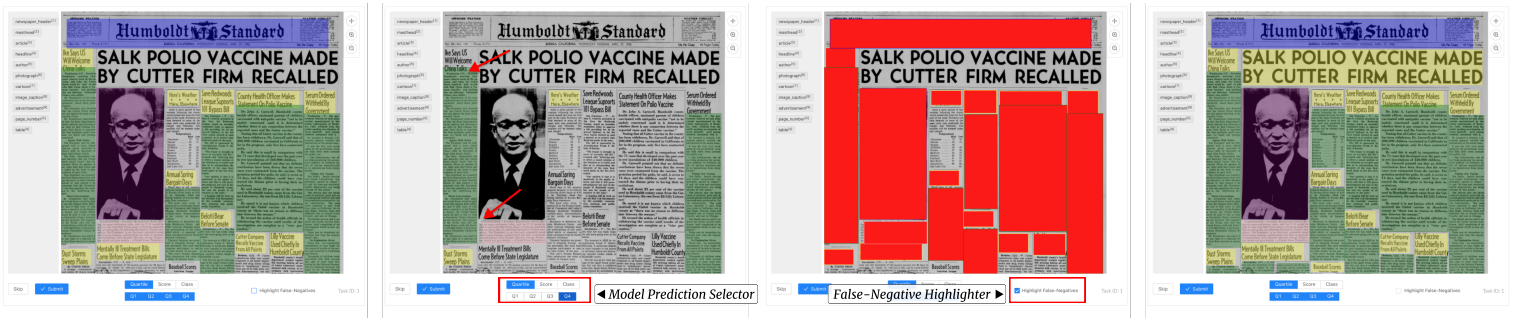
Shen et al., 2020

The method doesn't have a way to detect false negatives; humans must check for them or accuracy will implode.

In Practice

- Using OLALA in practice required integrating it, as well as other useful features, into an open source labeling software called Label Studio
- The software can be deployed locally or in the cloud

Labeling in Practice



(a) Full Model Predictions (b) Selected Model Predictions (c) Highlight False-Negative Regions (d) Final Created Annotation

Figure: Illustration of the annotation interface with OLALA features (Shen et all, 2020). (a) A pre-trained model generates object predictions, and they are highlighted as rectangular boxes on the original image. The color denotes the category. (b) The *Model Prediction Selector* enables hiding predictions of low object scores. In this case, objects of top 25% (the 4th Quartile, Q4) scores are presented. (c) The *False-Negative Highlighter* helps recognize mis-identified objects from the model predictions. After being enabled, it converts all predicted regions to a dummy color, and regions without predictions are highlighted.

Outline

Introduction

Active Learning for Layout Annotation

Labeling Hacks

Labeling Hacks: An Example

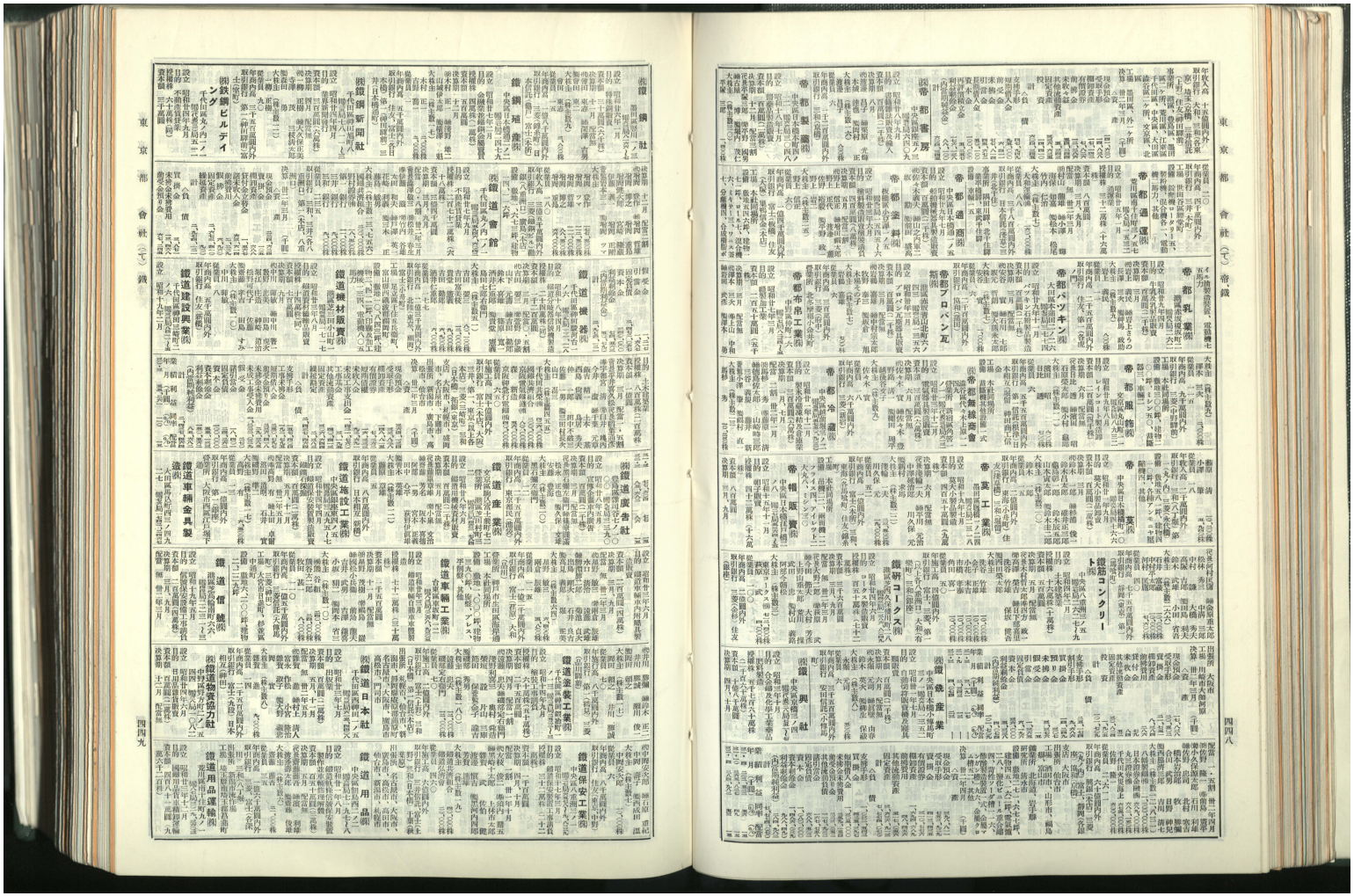
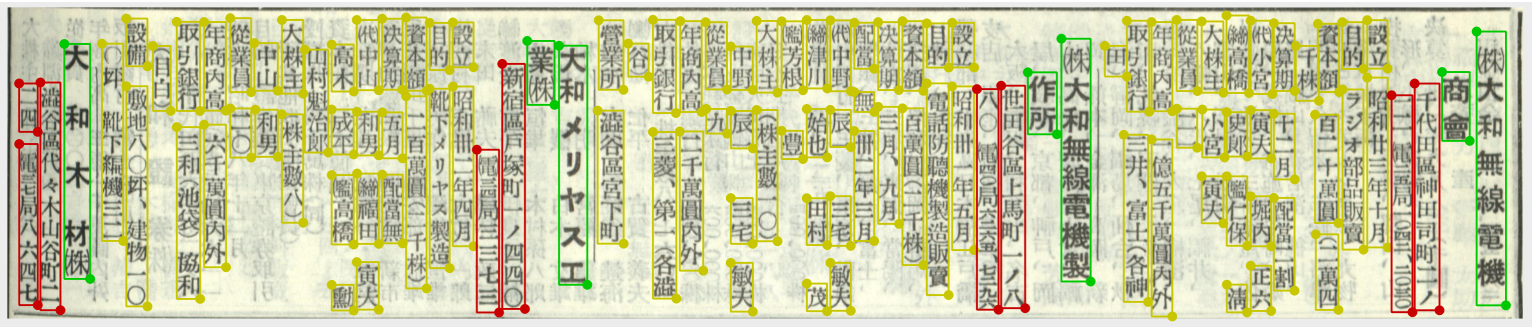


Figure: There are literally thousands of objects in this page scan. If you were to sit down and label this image, it would take hours to carefully label every object. You would need to *label every object in this image to use it for training*.

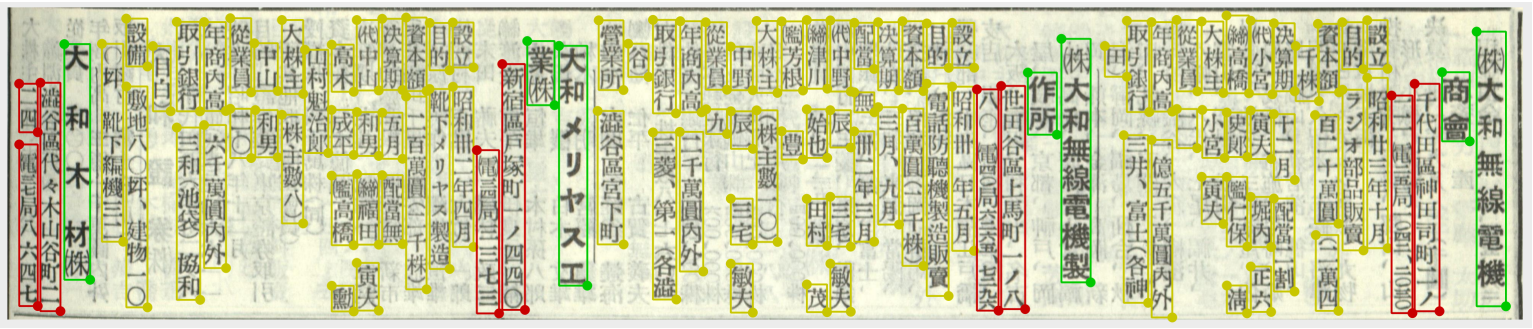
Labeling Hacks



Unpublished figure created by the author.

- We can make this problem more efficient by breaking the layouts down into their component pieces.
- To start, we segment this page into the sixteen rows that appear in the image. Still, each row has around 100-120 objects and obtaining enough samples to take a first pass at training the layout analysis model could be time-consuming.

Labeling Hacks



Unpublished figure created by the author.

- The average time to create a high-quality rectangular box for each object is around 20 seconds.
- Therefore, the average time for labeling one row is around 40 minutes. To label 60 row images would take 40 hours, a full work week.

Break It Down Further

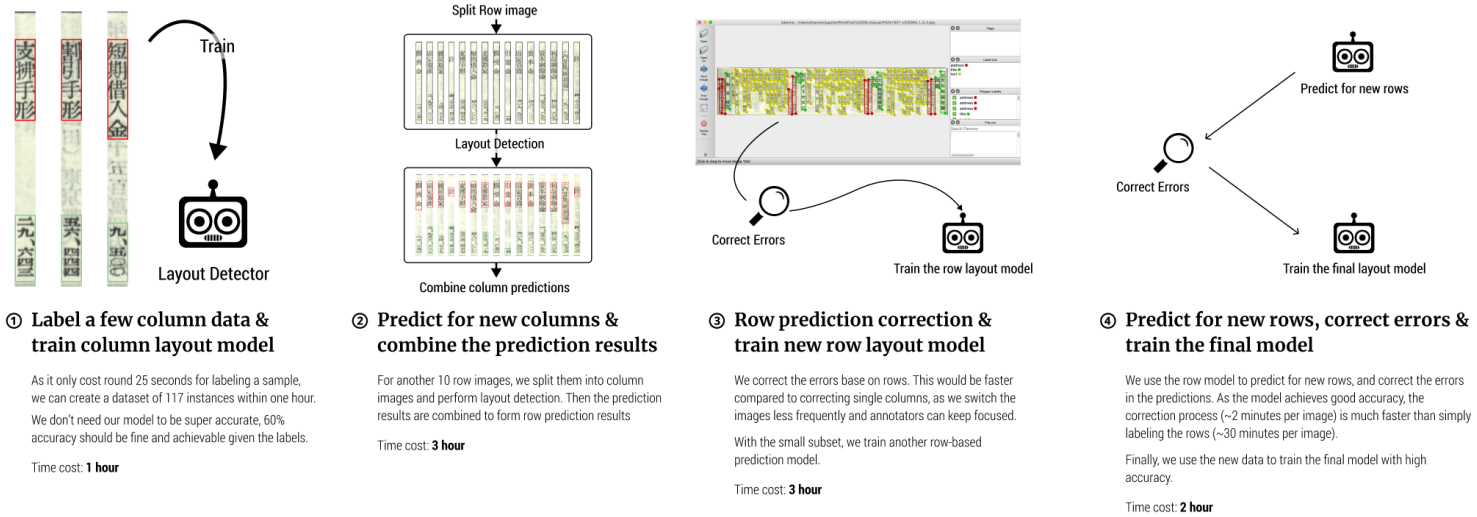


Figure: Breaking the layouts down further facilitates the initial labeling process.

Unpublished figure created by the author.

Step 1: Segment the rows into columns

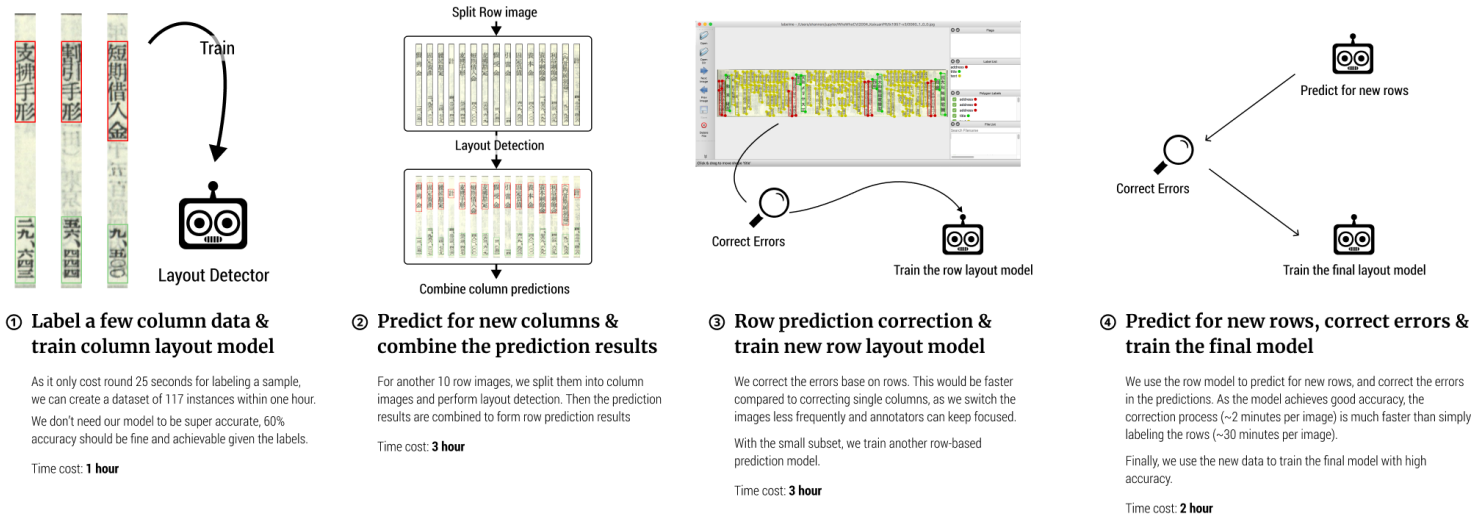


Figure: First we segment rows into columns. It is much faster to draw boxes around columns, and it does not take too many labeled samples to accurately segment the columns.

Unpublished figure created by the author.

Step 1: Segment the rows into columns

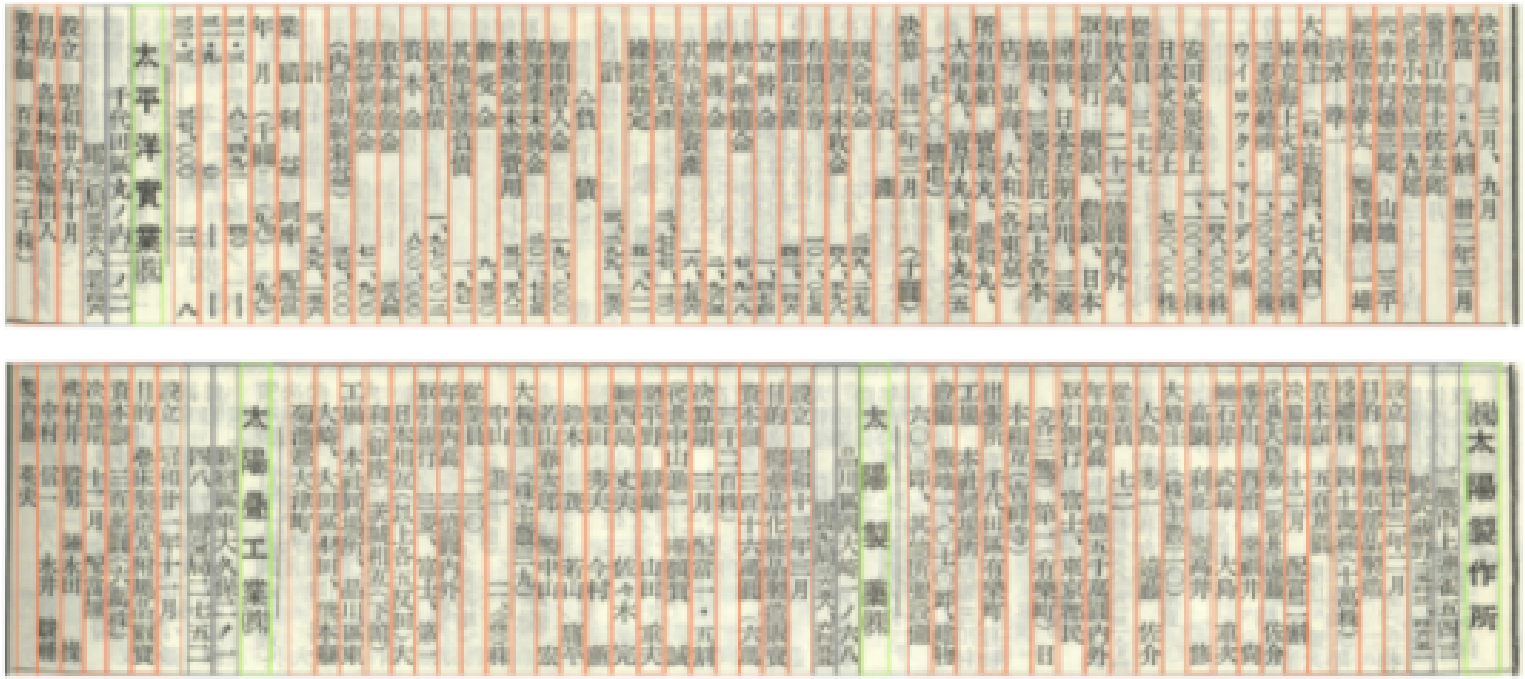


Figure: While labeling, we can also denote whether they are regular (red) or title columns (green), which look distinct. This can be useful - i.e., if the model is having a more difficult time accurately detecting objects in title columns, we can oversample row images with lots of title columns for labeling.

Unpublished figure created by the author.

Step 2: Predict layouts within columns

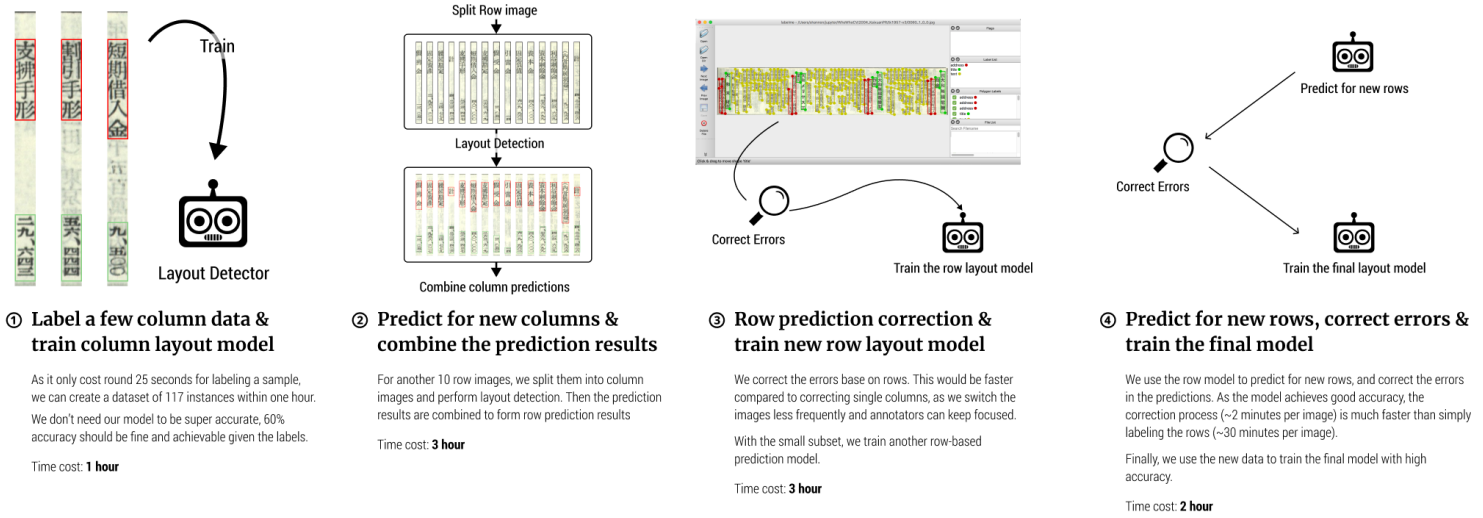


Figure: Next, we predict layout elements within individual columns, which have much simpler layouts than the full row image, with only 2-4 objects. Labeling a column often takes under a minute, so we can quickly label enough column samples to fine-tune the layout analysis model. We then merge the resulting column level predictions back into row level images to facilitate checking accuracy.

Unpublished figure created by the author.

Step 3: Correct labels and train a new row level prediction model

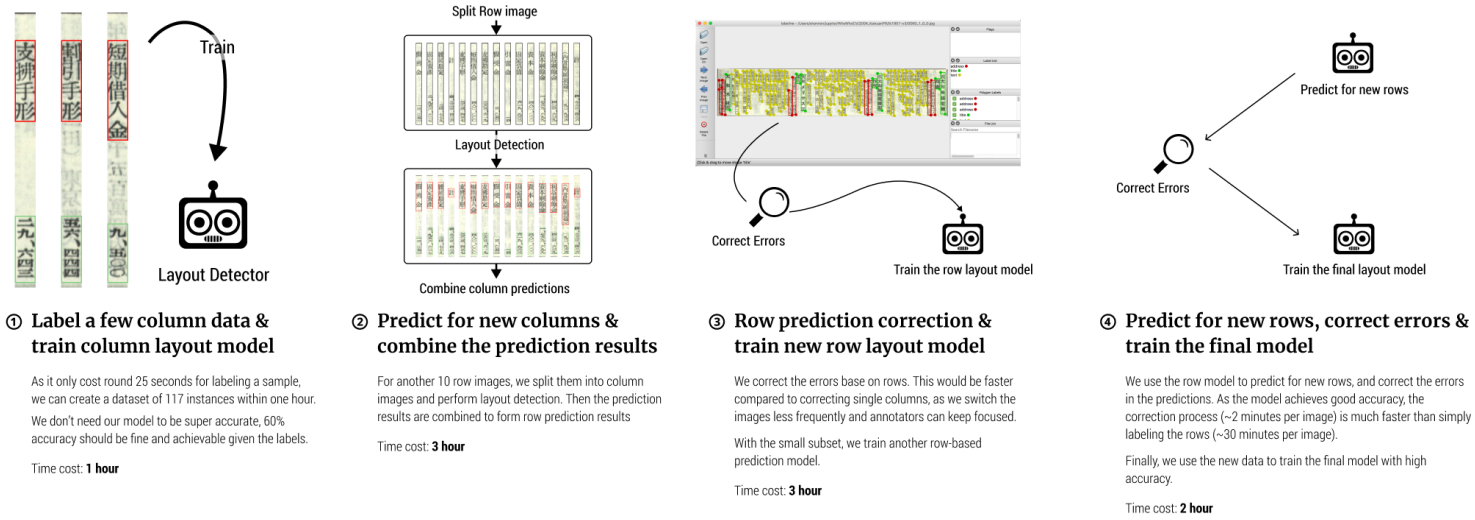


Figure: We correct the column level predictions after they have been merged back into a row level image and then use these row level labeled images to predict a layout model at the row level. Could integrate OLALA here.

Unpublished figure created by the author.

Step 4: Correct and Predict Until Acceptable Accuracy Achieved

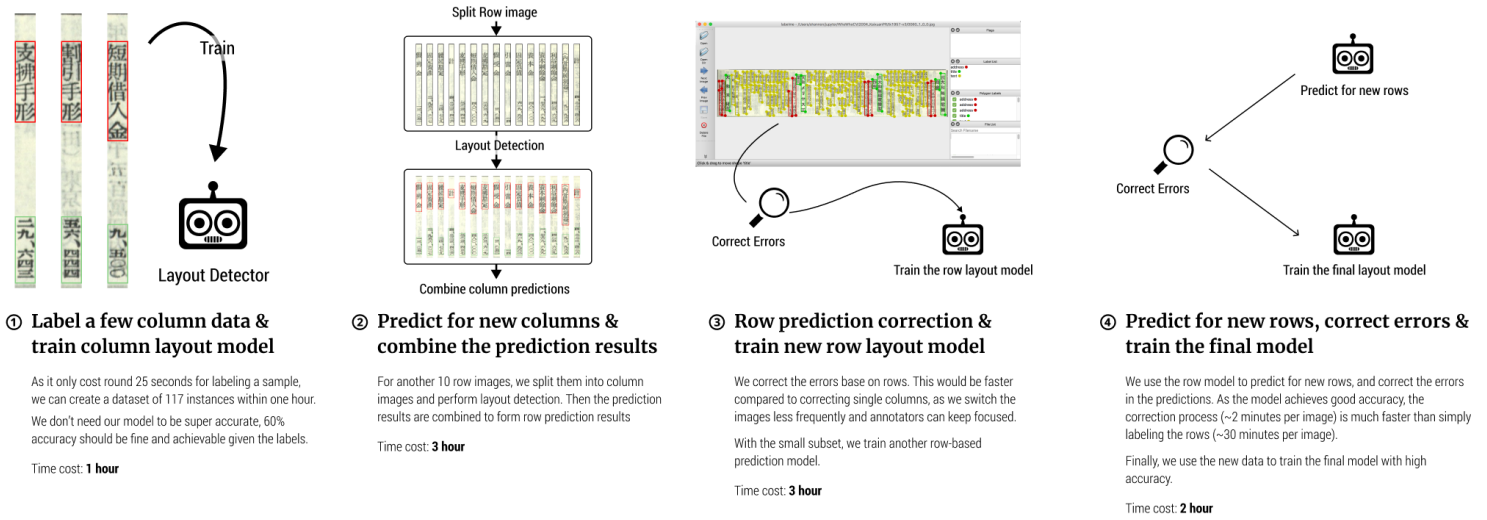


Figure: We correct errors in the predicted labels and use these labels to further train the model until an acceptable accuracy has been achieved. In this process, if certain predictions are less accurate, we can oversample rows with lots of predictions in that class.

Unpublished figure created by the author.