Supplementary Material for COMICS: More Details of Post-Processing and Dataset Creation

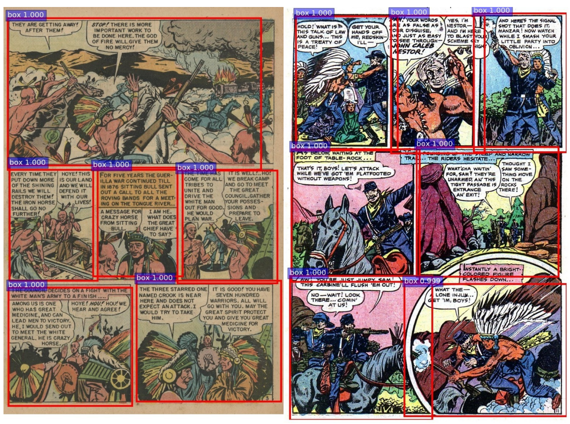


Figure 1. Two examples of panel-segmentation. On the left, we have a typical case where the panels are perfect rectangles, which results in perfect panel segmentation. Whereas on the right, panel layouts are atypical (the bottom-right panel is circular, and dialog-boxes spill over from one panel to another). Therefore, panel and subsequent textbox segmentation could possibly contain errors.

1. Panel detection using Faster R-CNN

Panels were detected from raw comic book page scans using Faster R-CNN 1. With regards to how we select the number of training examples used for training the R-CNNs, we observe that the shapes of panels (and textboxes) in our dataset follow a long-tailed distribution, with the majority of them being fairly standard (i.e., rectangular). Thus, only 500 manual annotations were sufficient to obtain robust panel detection. We ran an experiment on a small held out set of 20 random pages with 124 panels, and found the mean intersection-over-union overlap between ground-truth panel boxes and RCNN predicted boxes to be 0.911 (1.0 is perfect overlap).

2. OCR Post-Processing and Advertisement Removal

OCR makes systematic mistakes on our textboxes. We target two types of these mistakes using PyEnchant:[1] 1) where the OCR system fails to recognize the first letter of a particular word (e.g., *eleportation* instead of *teleportation*), and 2) where the OCR system transcribes part of a word as a single alphabetical character. To eliminate errors of the first type, we start by tokenizing the OCR output using NLTK's Punkt Tokenizer.[2] We then sort the vocabulary of the tokenized OCR output in decreasing order of frequency and pick words ranked from 10,001 to 100,000, because most misspelled words are also rare. For each of these words that is length three or longer, we look up the most likely suggestion offered by PyEnchant. If the only difference between the most likely suggestion and the original word is an additional letter in the first position of the suggestion, then we replace the word with the suggestion everywhere in our corpus. To correct the second type of errors, we simply delete all single character alphabetical tokens that are not one of 'a', 'd', 'i', 'm', 's', 't' - characters which can plausibly occur by themselves quite frequently (some occur after an apostrophe).

In addition to spelling errors, the books in COMICS contain many advertisements that we need to remove before generating data for our tasks. While most dialogue and narration boxes contain less than 30 words, longer textboxes frequently come from full-page product advertisements (e.g., Figure 2). However, detecting ads from page images is not easy. Some ads are deceptively similar to comic pages, containing images and even containing faux mini-comics. Aside from ads, there are also other undesirable pages; many books contain text-only short stories in addition to comics. We remove these kinds of pages using features from OCR transcriptions. We annotate each page of 100 random books with a label indicating the presence or absence of an invalid page as our training set and each page of 20 random books as our test set. Out of 6,117 annotated pages, 697 of them are either advertisements or text-only stories (11.4%). We train a binary classifier using Vowpal Wabbit:[3] which takes the OCR text for all the panels of a pages as lexical features (unigrams and bigrams). We improve our model by adding features like total count of words in the page and a count of non-alphanumeric char-

[1] http://pythonhosted.org/pyenchant/faq.html

[2] http://www.nltk.org/

[3] https://github.com/JohnLangford/vowpal_wabbit/wiki

acters. Our model gives us a total misclassification error of 8% and a false negative error of 17.3%, which means it misses one invalid page out of every six. The model has a negligible false positive error of 0.2%. Using this model to filter the entire dataset of 198,657 pages yields 13,200 invalid pages.

3. Examples from Dataset Creation

OCR transcription is the final stage of our data creation pipeline (panel extraction → textbox extraction → OCR). Therefore, faulty outputs in any of the preceeding steps can lead to faulty OCR outputs. In Figure 3, there are only minor errors in OCR extraction due to understandable misinterpretations of the text in the dialog boxes. For example, the OCR interprets the letters "IC" as "K", which leads to incorrectly predicting the word "QUICKLY" as "QUKKLY". However, in Figure 4, we observe a more critical error due to missing pixels in the panel extraction process. Due to the layout of the textbox in the panel, crucial portions of the text are trimmed from view; while the OCR does a valiant job of predicting the contents of the textbox, its output is gibberish.

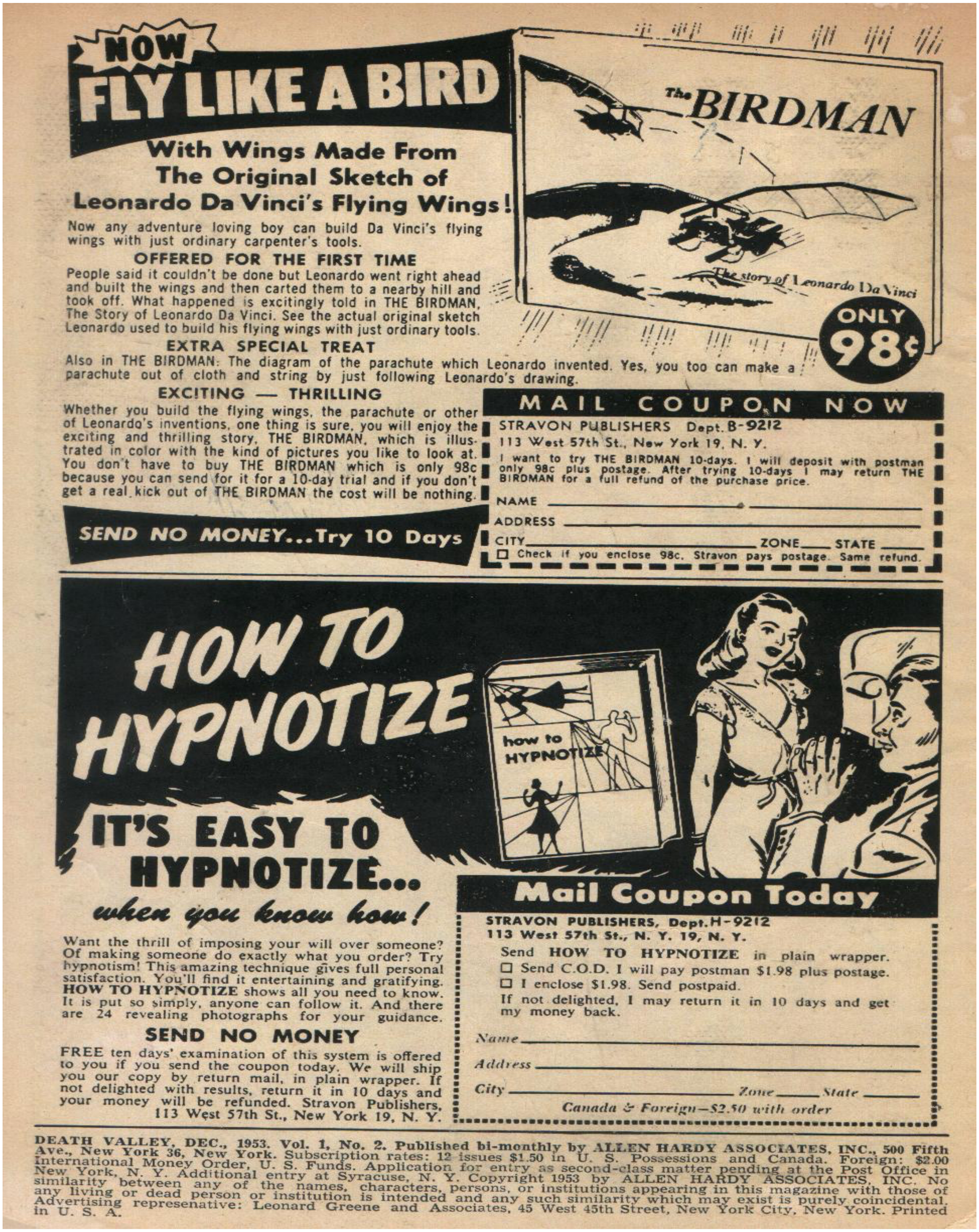
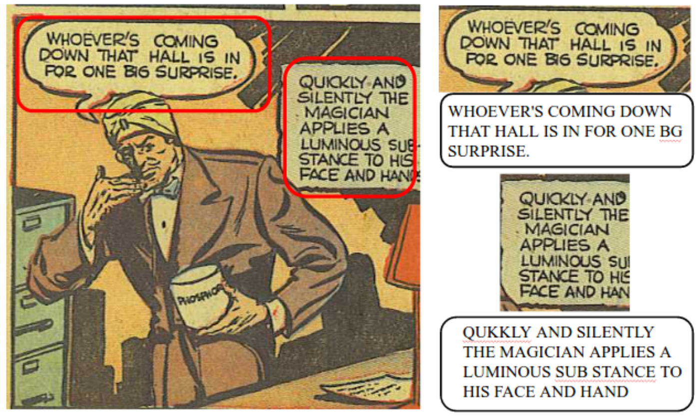
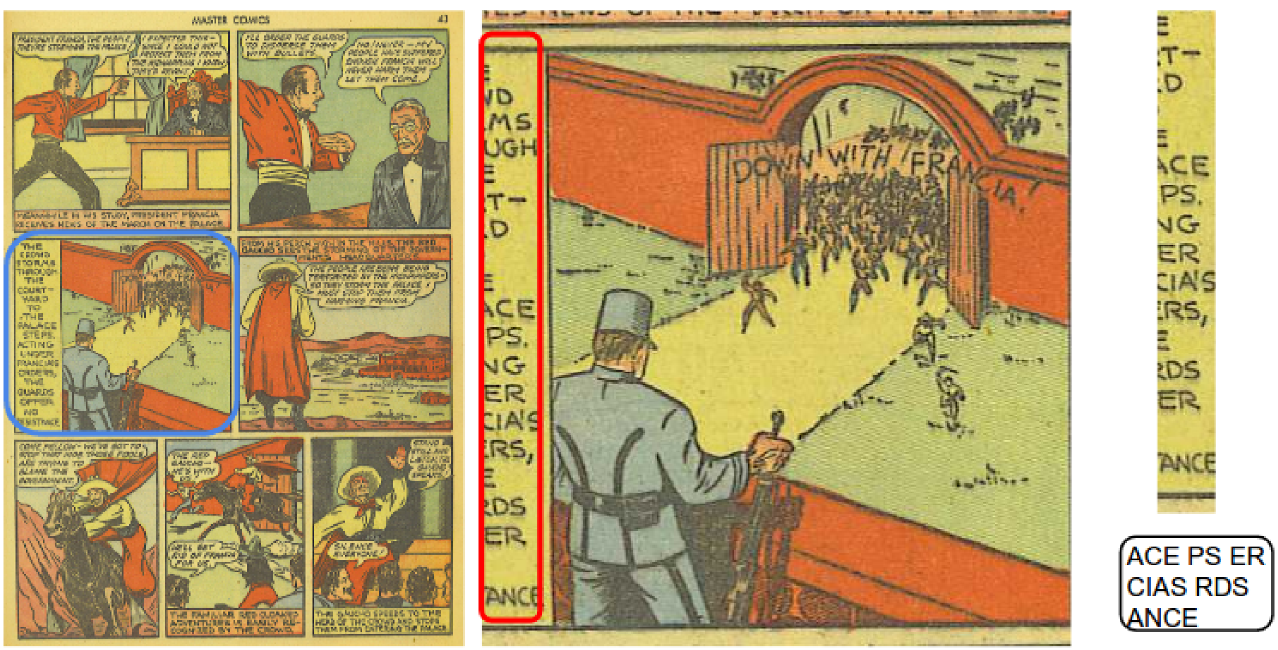


Figure 2. An advertisement from the dataset. The juxtaposition of text and image causes it to slightly resemble a comics page.



a. Output of Panel Extraction b. Textbox Extraction with OCR

Figure 3. A minor OCR error. Mistakes such as predicting "BG" for "BIG" are understandable, since the 'I' in "BIG" is barely visible. Similarly, the "IC" in "QUICKLY" looks a lot like "K" in this font. Finally, "SUB STANCE" is predicted rather than "SUBSTANCE", due to an end-of-line word break.



a. Page with ground truth panel location b. Output of Panel Extraction c. Textbox Extraction with OCR

Figure 4. A major OCR error. In part a) of the figure, note the location of the panel in the page. b) gives us the panel as predicted by the RCNN, but a critical portion of the text is missing. As a consequence, the textbox extraction is also faulty, rendering the OCR completely meaningless.