Project Work in Machine Learning for Computer Vision

Simone Persiani - 00984854

February 2023

1 Introduction

Source code for this project can be found at `https://github.com/iosonopersia/Document-Layout-Analysis`.

1.1 Problem definition

Document Layout Analysis is one of the main tasks of Document AI, a challenging research branch that refers to various techniques for accurately analyzing, organizing and parsing unstructured digital documents. The ability to efficiently and accurately gain insights about documents that are only human-readable (and not machine-readable) is highly requested in the business domain, as it could automate time-demanding and tedious tasks that are still assigned to humans, such as information retrieval from a big pile of resumes.

Documents can belong to many different domains and can be found in a variety of formats, such as forms, tables, receipts, invoices, tax forms, contracts, research papers, financial reports, books, ... Moreover, they can contain text written in any existing language, with different fonts, orientations, colours, sizes and so on.

Layout analysis can be thought of as a kind of Object Detection task where the model is asked to locate and classify regions of interest inside the image of a document page such as titles, text paragraphs, pictures, headers and footers, ... In the past years, many classical algorithms were designed to heuristically segment document images into their sub-parts. Lately, the advent of Vision Transformer models and of new techniques for self-supervised learning paved the way for a new research branch that promises better and better results.

1.2 Objective

The goal of my project is to test the efficacy of a novel backbone for Document AI on the Document Layout Analysis task. DiT [1] (Document Image Transformer) is a model based on ViT [2] (Vision Transformer) which, following BEiT [3], is pre-trained in a self-supervised way over a huge dataset of document images. This means that it can be exploited as a backbone inside the broader architecture of a complete object detection framework.

In particular, since the authors of DiT already tried fine-tuning their model using both the Mask R-CNN [4] and Cascade R-CNN [5] frameworks[1], I chose to implement an object detector based on the YOLOv3 [6] framework.

2 Dataset

The dataset that the authors of DiT used to evaluate performance on the Layout Analysis task is PubLayNet [7]. PubLayNet[2] is currently the largest annotated dataset for this kind of task, as it contains 358'353 images for a total number of 3'571'250 annotations.

Unfortunately, this dataset has some notable drawbacks:

- its size (∼102GB) makes it quite difficult to work with if using an average consumer desktop computer;
- it contains only images coming from scientific articles retrieved from PubMed Central[3], hence they span a relatively narrow domain (medical literature);
- it's not annotated by humans, as it was obtained by automatically matching the XML representations and the content of the original articles;
- following from the previous point, its annotations are assigned only to a coarse set of categories (namely *Text*, *Title*, *List*, *Table* and *Figure*) that may not be precise enough in many situations.

[1] Implementation available at `https://github.com/microsoft/unilm/tree/master/dit/object_detection`

[2] See `https://developer.ibm.com/data/publaynet/`

[3] See `https://www.ncbi.nlm.nih.gov/pmc/`

As a result, I chose to move to another dataset, even if this meant that my results wouldn't have been directly comparable to those obtained by DiT authors. Such dataset is called DocLayNet [8] and it's quite smaller in size ($\sim$35.5GB), which makes it easier to manipulate on a standard desktop computer. DocLayNet[4] contains 80'863 images for a total number of 1'107'470 annotations.

Unlike in PubLayNet, images are obtained from documents of vastly different domains (*scientific articles*, *financial reports*, *manuals*, *laws & regulations*, *patents* and *government tenders*), allowing for better generalization. The dataset is labelled by humans with the help of a comprehensive guidelines book[5] ($\sim$100 pages) and of a GUI software which helps to place bounding boxes with higher accuracy around ROIs. Finally, object categories were carefully chosen to maximise some quality metrics, resulting in a finer set of 11 classes (*Caption*, *Footnote*, *Formula*, *List-item*, *Page-footer*, *Page-header*, *Picture*, *Section-header*, *Table*, *Text*, and *Title*).

2.1 Data pipeline

Every image in DocLayNet is saved with the `.png` format, it's resized to 1025x1025 pixels and it's contained in the same `PNG` folder. The annotation files for the train, validation and test sets are stored as `.json` files inside the `COCO` folder, as they follow the COCO annotation format[6].

Two tasks are executed before starting the training process:

1. all the images from the training set are sequentially loaded to collect statistics about the mean and std of the normalized pixel values for each channel, with the following results (that are compatible with the fact that most pixels are just white background):

$$\mu = \begin{pmatrix} \mu_R \\ \mu_G \\ \mu_B \end{pmatrix} = \begin{pmatrix} 0.9328588400900901 \\ 0.9339698198198199 \\ 0.9338298423423423 \end{pmatrix}$$

$$\sigma = \begin{pmatrix} \sigma_R \\ \sigma_G \\ \sigma_B \end{pmatrix} = \begin{pmatrix} 0.1881268901781826 \\ 0.18381248888783014 \\ 0.18610593191303293 \end{pmatrix}$$

2. all the bounding box annotations from the training set are loaded and a K-Means clustering algorithm is run on their sizes (width and height) to select 12 centroids that will be used as anchors by the YOLOv3 head (results are shown in Figure 1). Following YOLOv2 [9], the distance metric used for clustering is the following:

$$d(box, centroid) = 1 - IoU(box, centroid)$$

In order to prepare the mini-batches for the training steps, first of all the right amount of images are loaded with pixel values belonging to the integer range of $[0, 255]$. Then, following DiT, the data augmentation technique from DETR [10] is applied to each of them, meaning that a random crop is performed with a probability of 50%.

3 Model

The architecture I implemented is the same which was used by DiT authors to fine-tune their model on PubLayNet: I tried to replicate it by using PyTorch [11] without using the Detectron2 framework [12] (as DiT authors did instead). The pre-trained parameters I used can be found at `https://huggingface.co/microsoft/dit-base`.

3.1 Input preparation

First of all, a batch of input images of size 1025x1025 pixels is transformed into the right input format for the Vision Transformer. Every image is:

- normalized to the decimal range of $[0.0, 1.0]$;
- standardized to a distribution $\sim N(0, 1)$ by means of the pre-computed μ and σ;
- resized to 224x224 pixels by means of a Bicubic interpolation;
- split into 14x14 patches of size 16x16 pixels.

[4] See `https://developer.ibm.com/data/doclaynet/`

[5] See `https://github.com/DS4SD/DocLayNet/blob/main/assets/DocLayNet_Labeling_Guide_Public.pdf`

[6] See `https://cocodataset.org/#format-data`

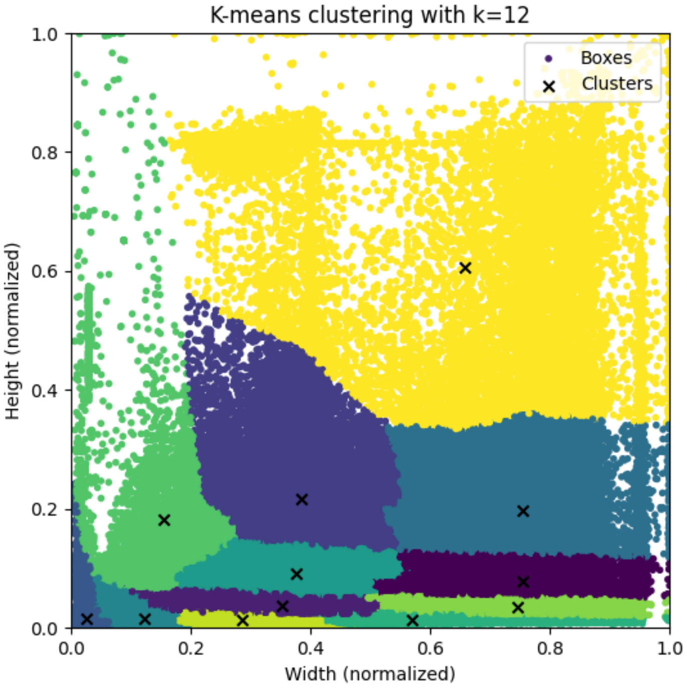


Figure 1: The 12 anchors chosen by executing the K-Means clustering algorithm on the training set annotations

The patches are flattened and projected to a latent space with exactly the hidden size used internally by the Transformer. Additionally, a `[CLS]` token is prefixed to the sequence of patches and an absolute positional encoding is summed to each of them so that the model can infer their relative positions inside the original image.

3.2 Feature maps extraction

Because of hardware limitations, I was forced to use the "base" variant of DiT, which has 12 encoder blocks, a hidden size of 768 dimensions and 12 self-attention heads for each encoder.

Following DiT, I extract hidden outputs at the following layers:

- 4^{th}, which is upscaled from 14x14 to 56x56 by means of 2 subsequent 2x2 transposed convolutions with stride 2 (with an in-between Batch Norm with GELU activation);
- 6^{th}, which is upscaled from 14x14 to 28x28 by means of a single 2x2 transposed convolution with stride 2;
- 8^{th}, which is left as it is (14x14 resolution) without any additional transformations;
- 12^{th}, which is downscaled from 14x14 to 7x7 by means of a 2x2 max pooling with stride 2.

3.3 FPN and YOLO head

All those feature maps are then passed as input to a Feature Pyramid Network [13] which, through a top-down path, brings high-level semantic features from the deeper feature maps to the shallower ones. The number of channels computed for each grid is 256.

Finally, each of them is fed to the YOLO head: a 1x1 convolution which outputs $3 \cdot (5 + C)$ channels, where 3 is the number of predicted anchors for each cell, C is the number of classes (11 in DocLayNet) and 5 is used to account both for the objectness score and for the box coordinates.

4 Experiments

4.1 Debugging and overfit test

At first, I developed a dummy model and a dummy dataset which helped me a lot for debugging purposes.

In the dummy model, the Transformer-based backbone is replaced by a very simple 3-layers CNN that produces a single 14x14 feature map. The latter is then rescaled as in the real model to produce 4 maps of different sizes, which are then fed to the FPN and finally to the YOLO head. By selectively disabling one of these parts at a time, I was able to quickly discover whether any of them was the cause of some problems or not.

On the other hand, the dummy dataset is a simple class which is capable of procedurally generating a highly simplified set of 11 images (one for each class in DocLayNet) for which the target is trivial to predict. In particular, each image is built starting from a white background of size 1025x1025 pixels and drawing a 200x200 pixels square at the center. The colour of the square is a greyscale value which is directly proportional to the class label index. This simplifies a lot the object detection task as there's always just one object in the same position and with the same dimensions, with a colour that allows to easily infer the class it belongs to.

I used the dummy dataset to perform some early overfit tests with the dummy model. Once I fixed some bugs related to other parts of the project (mainly coming from the implementation of the YOLOv3 loss function) and I verified that everything was working properly, I also tried using the real model instead of the dummy one. As expected, even this overfit test was passed successfully.

4.2 Training settings

I trained the model on my personal desktop computer, which features an `AMD R5 3600` CPU and a `NVIDIA GTX 1660 Super` GPU. Since the on-board GPU memory is only 6GB in size, I was forced to select the "base" version of DiT (with ∼87M parameters instead of the ∼304M of the "large" variant). This choice shouldn't have had a big impact on my results, as the reported performance difference between the two variants on the PubLayNet dataset is only marginal (93.5% COCO mAP instead of 94.1% with the "large" variant).

The model was trained for 8 epochs with a mini-batch size of 8 samples, for a total of 8'637 update steps per epoch. DiT authors used mini-batches of 16 samples, but I had to find a compromise because of the aforementioned lack of GPU memory. Learning rate was linearly increased from 0 to $2e-4$ during the first epoch and then slowly reduced back to 0 thanks to a Cosine scheduler. The AdamW optimizer was used with a weight decay factor of 0.05. I also applied gradient clipping with a maximum Euclidean norm of 1.0. Finally, I used an early stopping strategy on the validation loss with a patience of 3 epochs.

The only difference with what was done by DiT authors on PubLayNet is that they used a learning rate of $4e-4$, while I found a lower value to bring better results on DocLayNet. Moreover, their warmup schedule starts from $4e-6$ (instead of 0) and it's only 1'000 iterations long ($\sim \frac{1}{8}$ of an epoch of mine).

Notably, the number of epochs was chosen to match (with some additional margin) the number of iterations set by DiT authors in the Detectron2 `YAML` config file, which is 60'000 ($8 \cdot 8637 =$ 69'096).

4.3 Results

In order to be able to evaluate the results of my model on the DocLayNet test set, I had to implement from scratch two important algorithms: the one which performs Non-Maximum Suppression for each object category separately and the one which computes the COCO mAP metric, namely the mAP@0.5:0.95. I also implemented a simple visualization function which plots both the original image and the predicted bounding boxes with their corresponding labels, for an easier inspection on the outputs of the model at evaluation time.

The whole run took a total of ∼ 11 hours and a half, as the early stopping strategy was never triggered and both training and validation loss always decreased. This suggests that I really should have trained the model for a longer time, which unfortunately wasn't possible in my case.

Accordingly, the overall COCO mAP computed on the test set of DocLayNet is only of ∼ 24.5%. Such figure is obviously way too low: this is mostly due to the mAP values computed at the highest IoU thresholds that are in the order of magnitude of 1%. By comparison, DocLayNet authors were able to obtain a much higher result of 76.8% with an inter-annotator agreement of ∼ 82-83%, by adopting the YOLOv5 v6.0 implementation by Ultralytics [14]. A complete list of such results is given in Table 1. Please note that the mAP computation is performed after a NMS pass with a minimum objectness confidence of 0.1 and an IoU threshold of 0.45.

5 Conclusions

5.1 Comments on the results

Summing up, I achieved the goal of implementing a full architecture for object detection in document images, which is based on a pre-trained Vision Transformer backbone, a FPN and a YOLOv3 head. Unfortunately, due to hardware and time limitations, I wasn't able to fully train the model and this fact is made evident by the poor results I got on the test set of DocLayNet.

Actually the overall mAP, even if it starts as high as 43.6%, drops quite rapidly as the IoU threshold grows, with very low results reached for mAP@0.90 and mAP@0.95. Category-wise, results are low everywhere, with the lowest result obtained for the *List-item* class.

Category	**mAP (%) at each IoU threshold**										**Overall**
	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95	
Caption	40.0	37.0	33.7	30.2	26.9	22.9	18.8	15.0	8.2	1.6	**23.4**
Footnote	45.4	43.7	40.9	37.3	33.2	27.0	21.4	11.9	4.1	0.2	**26.5**
Formula	28.7	25.8	22.6	19.2	14.0	8.3	4.9	1.3	0.1	0.0	**12.5**
List-item	16.5	15.2	14.2	12.9	11.6	10.1	8.1	5.5	2.4	0.2	**9.7**
Page-footer	76.2	71.4	65.4	57.8	48.7	35.8	21.5	10.0	2.6	0.0	**38.9**
Page-header	56.9	53.5	49.3	43.6	37.1	31.4	25.3	16.9	6.0	0.3	**32.0**
Picture	42.2	39.9	37.5	34.8	31.9	27.7	22.2	13.9	3.8	0.1	**25.4**
Section-header	40.3	37.5	34.2	29.2	22.4	13.5	6.5	2.4	0.4	0.0	**18.6**
Table	61.3	59.3	57.2	54.3	51.5	46.8	39.8	28.6	11.7	0.7	**41.1**
Text	25.7	24.2	22.8	21.3	19.9	18.0	16.0	13.2	7.9	1.0	**17.0**
Title	46.8	41.3	38.1	35.0	31.1	23.2	17.7	9.6	3.3	0.1	**24.6**
Overall	**43.6**	**40.8**	**37.8**	**34.1**	**29.8**	**24.1**	**18.4**	**11.7**	**4.6**	**0.4**	24.5

Table 1: mAP results (%) rounded to the closest first decimal digit

These results suggest that the model is incapable of producing a precise localization of the bounding boxes and that it performs better on some classes with respect to others. Notably, the 3 most frequent classes in the dataset (*Text*, *List-item* and *Section-header*, which represent respectively 45.82%, 17.19% and 12.6% of the training set) perform quite worse than much more infrequent classes such as *Page-footer* (6.51% of the training set) and *Table* (3.2% of the training set).

It's possible that DocLayNet, being a much more general dataset than PubLayNet, could require more training iterations to achieve its best results. In fact, even if it is $\sim$ 4.5 times bigger in size (with respect to the total number of images), PubLayNet only contains samples from medical scientific articles, which can help quite a lot to achieve better results with the same amount of training steps.

5.2 Future work

A future work should certainly try to train again this architecture for a longer time, in order to determine whether the low results I obtained are only due to the training process being interrupted too early or because of other factors (maybe even implementation errors).

It could also be interesting to try experimenting with the following techniques:

- label smoothing for the BCE losses used for the objectness score and the class logits, for additional regularization;
- a proper initialization of the biases in the 1x1 convolution of the YOLO head by choosing appropriate values based on the relative frequency of each class, so to make training loss converge faster during the initial iterations of the first epoch;
- introduction of a GELU activation on the pooled outputs from DiT hidden layers;
- introduction of a normalization layer and a GELU activation on top of the rescaling network for each feature map before it's fed to the FPN, for a faster convergence during the initial training steps;
- introduction of normalization layers inside the FPN, for a faster convergence during the initial training steps;
- usage of the Focal Loss, which may be beneficial even if the author of YOLOv3 stated otherwise;
- class balancing by using proper weights in the BCE losses used for the objectness score and the class logits;
- trying to apply weighing coefficients to each term of the YOLOv3 loss (for example, to speed up convergence of the MSE loss for the bounding box predictions, which appears to be the main Achille's heel of my experiment);
- finally, more modern object detection frameworks could be employed, such as CenterNet or later versions of YOLO.

References

[1] Junlong Li et al. *DiT: Self-supervised Pre-training for Document Image Transformer*. 2022. DOI: 10.48550/ARXIV.2203.02378. URL: https://arxiv.org/abs/2203.02378.

[2] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: 10.48550/ARXIV.2010.11929. URL: https://arxiv.org/abs/2010.11929.

[3] Hangbo Bao et al. *BEiT: BERT Pre-Training of Image Transformers*. 2021. DOI: 10.48550/ARXIV.2106.08254. URL: https://arxiv.org/abs/2106.08254.

[4] Kaiming He et al. *Mask R-CNN*. 2017. DOI: 10.48550/ARXIV.1703.06870. URL: https://arxiv.org/abs/1703.06870.

[5] Zhaowei Cai and Nuno Vasconcelos. *Cascade R-CNN: Delving into High Quality Object Detection*. 2017. DOI: 10.48550/ARXIV.1712.00726. URL: https://arxiv.org/abs/1712.00726.

[6] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. DOI: 10.48550/ARXIV.1804.02767. URL: https://arxiv.org/abs/1804.02767.

[7] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. *PubLayNet: largest dataset ever for document layout analysis*. 2019. DOI: 10.48550/ARXIV.1908.07836. URL: https://arxiv.org/abs/1908.07836.

[8] Birgit Pfitzmann et al. “DocLayNet: A Large Human-Annotated Dataset for Document-Layout Segmentation”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2022. DOI: 10.1145/3534678.3539043. URL: https://doi.org/10.1145/3534678.3539043.

[9] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. DOI: 10.48550/ARXIV.1612.08242. URL: https://arxiv.org/abs/1612.08242.

[10] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. DOI: 10.48550/ARXIV.2005.12872. URL: https://arxiv.org/abs/2005.12872.

[11] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. DOI: 10.48550/ARXIV.1912.01703. URL: https://arxiv.org/abs/1912.01703.

[12] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[13] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2016. DOI: 10.48550/ARXIV.1612.03144. URL: https://arxiv.org/abs/1612.03144.

[14] Glenn Jocher et al. *ultralytics/yolov5: v6.0 - YOLOv5n ’Nano’ models, Roboflow integration, TensorFlow export, OpenCV DNN support*. Version v6.0. Oct. 2021. DOI: 10.5281/zenodo.5563715. URL: https://doi.org/10.5281/zenodo.5563715.

Appendix

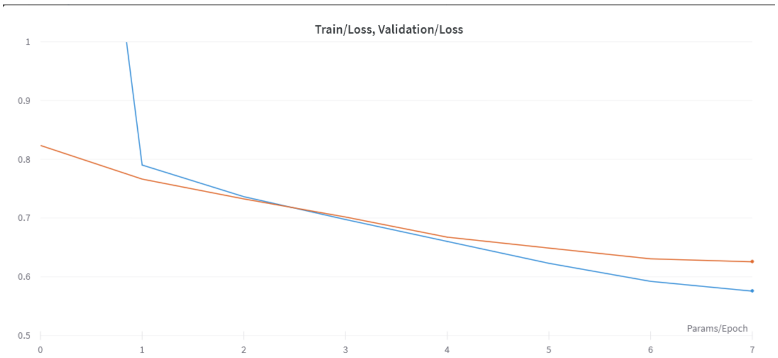


Figure 2: Training and validation loss at each epoch (the blue and the orange lines, respectively)

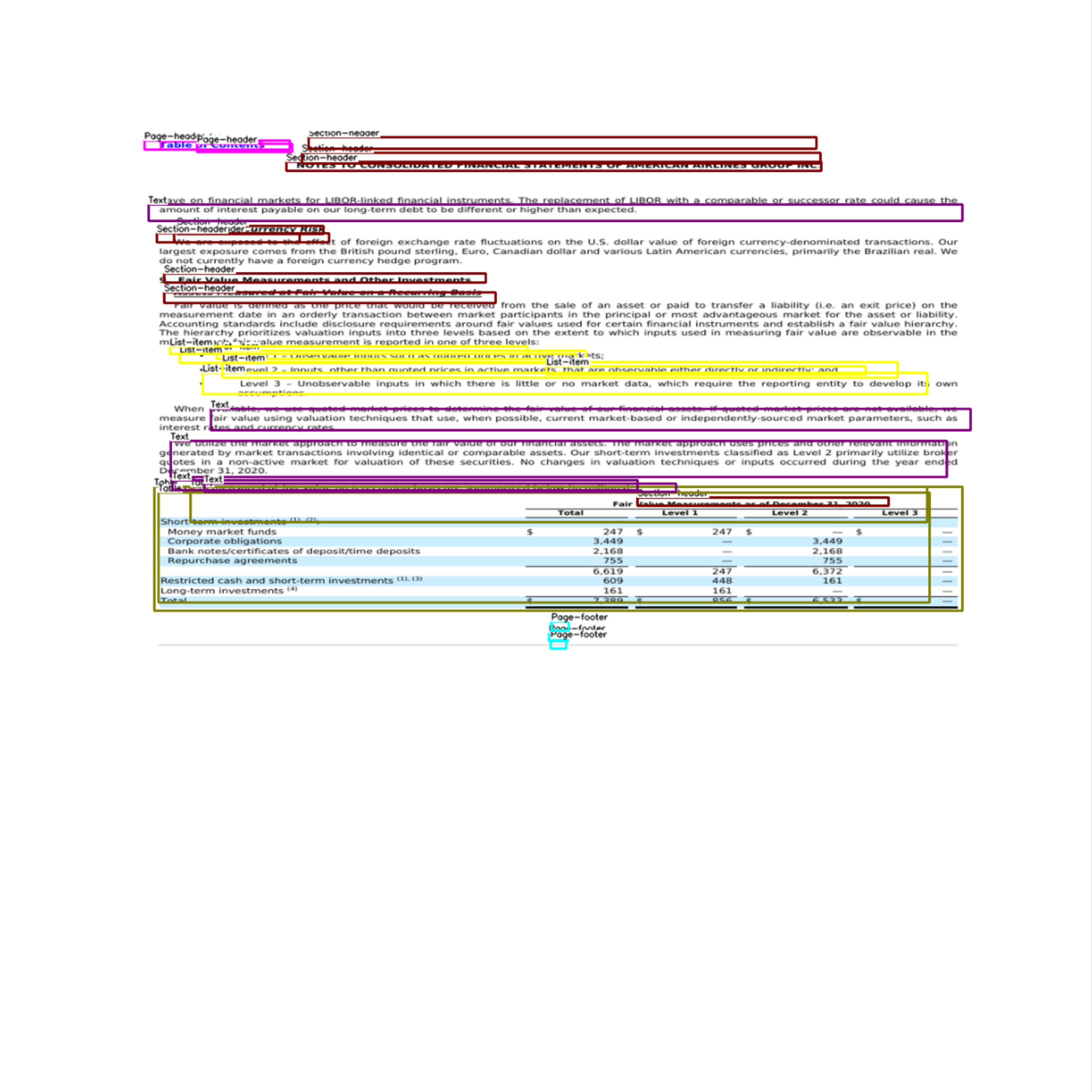


Table of Contents

NOTES TO CONSOLIDATED FINANCIAL STATEMENTS OF AMERICAN AIRLINES GROUP INC.

ave on financial markets for LIBOR-linked financial instruments. The replacement of LIBOR with a comparable or successor rate could cause the amount of interest payable on our long-term debt to be different or higher than expected.

Currency Risk

We are exposed to the effect of foreign exchange rate fluctuations on the U.S. dollar value of foreign currency-denominated transactions. Our largest exposure comes from the British pound sterling, Euro, Canadian dollar and various Latin American currencies, primarily the Brazilian real. We do not currently have a foreign currency hedge program.

Fair Value Measurements and Other Investments

Assets Measured at Fair Value on a Recurring Basis

Fair value is defined as the price that would be received from the sale of an asset or paid to transfer a liability (i.e. an exit price) on the measurement date in an orderly transaction between market participants in the principal or most advantageous market for the asset or liability. Accounting standards include disclosure requirements around fair values used for certain financial instruments and establish a fair value hierarchy. The hierarchy prioritizes valuation inputs into three levels based on the extent to which inputs used in measuring fair value are observable in the m[illegible] fair value measurement is reported in one of three levels:

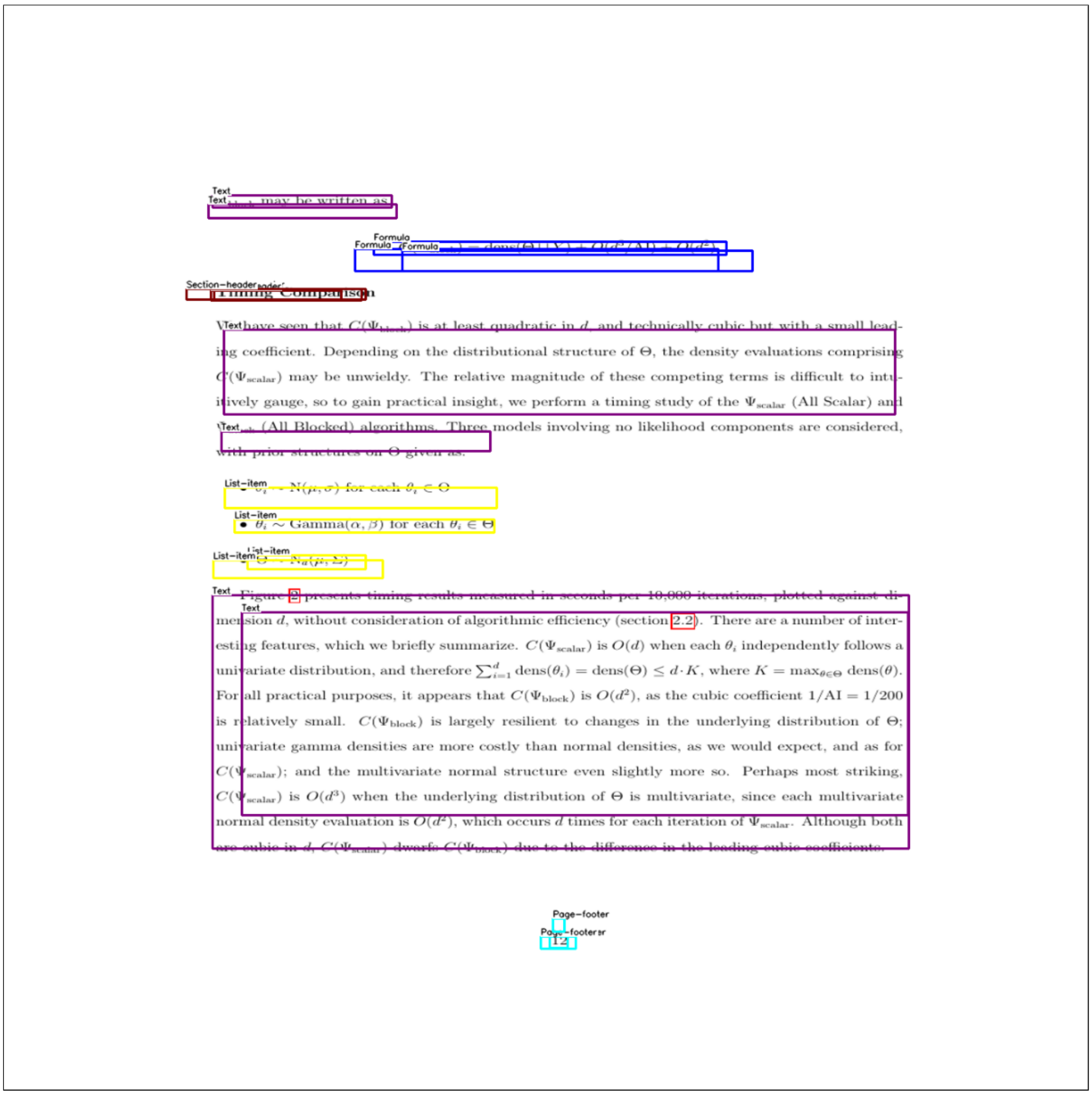
- Level 1 – Observable inputs such as quoted prices in active markets;
- Level 2 – Inputs, other than quoted prices in active markets, that are observable either directly or indirectly; and
- Level 3 – Unobservable inputs in which there is little or no market data, which require the reporting entity to develop its own assumptions.

When available, we use quoted market prices to determine the fair value of our financial assets. If quoted market prices are not available, we measure fair value using valuation techniques that use, when possible, current market-based or independently-sourced market parameters, such as interest rates and currency rates.

We utilize the market approach to measure the fair value of our financial assets. The market approach uses prices and other relevant information generated by market transactions involving identical or comparable assets. Our short-term investments classified as Level 2 primarily utilize broker quotes in a non-active market for valuation of these securities. No changes in valuation techniques or inputs occurred during the year ended December 31, 2020.

	Fair Value Measurements as of December 31, 2020			
	Total	Level 1	Level 2	Level 3
Short-term investments [(1), (2)]:				
Money market funds	$ 247	$ 247	$ —	$ —
Corporate obligations	3,449	—	3,449	—
Bank notes/certificates of deposit/time deposits	2,168	—	2,168	—
Repurchase agreements	755	—	755	—
	6,619	247	6,372	—
Restricted cash and short-term investments [(1), (3)]	609	448	161	—
Long-term investments [(4)]	161	161	—	—
Total	$ 7,389	$ 856	$ 6,533	$ —

Figure 3: An example of inference made on a test image of type *Financial report*



Ψ_{block} may be written as

$$C(\Psi_{\text{block}}) = \text{dens}(\Theta \mid Y) + O(d^3/\text{AI}) + O(d^2)$$

Timing Comparison

We have seen that $C(\Psi_{\text{block}})$ is at least quadratic in d, and technically cubic but with a small leading coefficient. Depending on the distributional structure of Θ, the density evaluations comprising $C(\Psi_{\text{scalar}})$ may be unwieldy. The relative magnitude of these competing terms is difficult to intuitively gauge, so to gain practical insight, we perform a timing study of the Ψ_{scalar} (All Scalar) and Ψ_{block} (All Blocked) algorithms. Three models involving no likelihood components are considered, with prior structures on Θ given as:

- $\theta_i \sim \text{N}(\mu, \sigma)$ for each $\theta_i \in \Theta$
- $\theta_i \sim \text{Gamma}(\alpha, \beta)$ for each $\theta_i \in \Theta$
- $\Theta \sim \text{N}_d(\mu, \Sigma)$

Figure 2 presents timing results measured in seconds per 10,000 iterations, plotted against dimension d, without consideration of algorithmic efficiency (section 2.2). There are a number of interesting features, which we briefly summarize. $C(\Psi_{\text{scalar}})$ is $O(d)$ when each θ_i independently follows a univariate distribution, and therefore $\sum_{i=1}^{d} \text{dens}(\theta_i) = \text{dens}(\Theta) \leq d \cdot K$, where $K = \max_{\theta \in \Theta} \text{dens}(\theta)$. For all practical purposes, it appears that $C(\Psi_{\text{block}})$ is $O(d^2)$, as the cubic coefficient $1/\text{AI} = 1/200$ is relatively small. $C(\Psi_{\text{block}})$ is largely resilient to changes in the underlying distribution of Θ; univariate gamma densities are more costly than normal densities, as we would expect, and as for $C(\Psi_{\text{scalar}})$; and the multivariate normal structure even slightly more so. Perhaps most striking, $C(\Psi_{\text{scalar}})$ is $O(d^3)$ when the underlying distribution of Θ is multivariate, since each multivariate normal density evaluation is $O(d^2)$, which occurs d times for each iteration of Ψ_{scalar}. Although both are cubic in d, $C(\Psi_{\text{scalar}})$ dwarfs $C(\Psi_{\text{block}})$ due to the difference in the leading cubic coefficients.

12

Figure 4: An example of inference made on a test image of type *Scientific article*