Figure Extraction from Scanned Electronic Theses and Dissertations

Sampanna Yashwant Kahu

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Edward A. Fox, Chair
A. Lynn Abbott
William Diehl

August 6, 2020
Blacksburg, Virginia

Keywords: Figure Extraction, Deep Learning, Computer Vision, Digital Libraries

Figure Extraction from Scanned Electronic Theses and Dissertations

Sampanna Yashwant Kahu

(ABSTRACT)

The ability to extract figures and tables from scientific documents can solve key use-cases such as their semantic parsing, summarization, or indexing. Although a few methods have been developed to extract figures and tables from scientific documents, their performance on scanned counterparts is considerably lower than on born-digital ones. To facilitate this, we propose methods to effectively extract figures and tables from Electronic Theses and Dissertations (ETDs), that out-perform existing methods by a considerable margin. Our contribution towards this goal is three-fold. (a) We propose a system/model for improving the performance of existing methods on scanned scientific documents for figure and table extraction. (b) We release a new dataset containing 10,182 labelled page-images spanning across 70 scanned ETDs with 3.3k manually annotated bounding boxes for figures and tables. (c) Lastly, we release our entire code and the trained model weights to enable further research (https://github.com/SampannaKahu/deepfigures-open).

Figure Extraction from Scanned Electronic Theses and Dissertations

Sampanna Yashwant Kahu

(GENERAL AUDIENCE ABSTRACT)

Portable Document Format (PDF) is one of the most popular document formats. However, parsing PDF files is not a trivial task. One use-case of parsing PDF files is the search functionality on websites hosting scholarly documents (i.e., IEEE Xplore, etc.). Having the ability to extract figures and tables from a scholarly document helps this use-case, among others. Methods using deep learning exist which extract figures from scholarly documents. However, a large number of scholarly documents, especially the ones published before the advent of computers, have been scanned from hard paper copies into PDF. In particular, we focus on scanned PDF versions of long documents, such as Electronic Theses and Dissertations (ETDs). No experiments have been done yet that evaluate the efficacy of the above-mentioned methods on this scanned corpus. This work explores and attempts to improve the performance of these existing methods on scanned ETDs. A new gold standard dataset is created and released as a part of this work for figure extraction from scanned ETDs. Finally, the entire source code and trained model weights are made open-source to aid further research in this field.

Dedication

To my family

Acknowledgments

First and foremost, I thank the Almighty for blessing me with the knowledge and skills required to accomplish the thesis.

I would like to thank Dr. Edward A. Fox for providing me the opportunity to work on this project and for his advice on academics and career. His constant guidance and support have made my journey through the Master's degree a rewarding one. Completing the thesis on time would not have been possible without his prompt feedback.

I would also like to thank my committee members, Dr. A. Lynn Abbott and Dr. William Diehl.

Many thanks to William Ingram for the countless meetings and discussions we had during the course of our classwork and my research, which ultimately culminated into this thesis.

I would like to thank Dr. Jian Wu, Assistant Professor, Old Dominion University for his invaluable inputs.

Further, I would also like to thank to Dr. Waleed Ammar, Allen AI for his helpful suggestions over e-mail.

I thank my parents for relentlessly supporting my decision to pursue a Master's degree. I thank my sister, Samruddhi Kahu-Talegaonkar and brother-in-law, Mrunmay Talegaonkar for the constant and invaluable guidance they provided in this wonderful journey.

Finally, my gratitude to all my friends, colleagues and well-wishers without whom this work wouldn't have been possible.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

APDFL Adobe PDF Library

ARC Advanced Research Computing

arXiv Open access archive in science and scholarly fields run by Cornell University [1]

CNN Convolutional Neural Network

CPU Central Processing Unit

DLRL Digital Library Research Laboratory

ETDs Electronic Theses and Dissertations

FN False Negative

FP False Positive

GB Gigabyte

GPU Graphics Processing Unit

GUI Graphical User Interface

HPC High Performance Computing

IEEE Institute of Electrical and Electronics Engineers

IOU Intersection Over Union

[1] https://arXiv.org

JSON JavaScript Object Notation

LaTeX A document preparation system, especially for mathematical documents

MB Megabyte

ML Machine Learning

PDF Portable Document Format

POC Proof Of Concept

ResNet Residual Networks [33]

RQ Research Question

TB Terabyte

TEI Text Encoding Initiative

TP True Positive

XML Extensible Markup Language

Chapter 1

Introduction

1.1 Motivation

Figures and tables in scholarly papers contain important information. Automatic identification and extraction of figures and tables from PDF documents is key to enhancing computational access to scholarly works[1].

Our research focuses on Electronic Theses and Dissertations (ETDs). Beginning with Virginia Tech in 1997 and continuing today, graduate programs all over the world allow (or often mandate) electronic submission of an ETD as a requirement for graduation. University libraries often provide public access to digital libraries of ETDs. Some universities scan or digitize older theses and dissertations, in order to provide electronic access to these works. For instance, Virginia Tech's collection of ETDs now goes back to the year 1903. Most ETDs written before the late 1990s are scanned versions of the library's physical collection of theses and dissertations. Our work aims to identify and extract figures from these older, scanned documents.

There are many engineering challenges to accurately identifying figures in older scanned ETDs. The image resolution and scanning quality may vary across the collection. OCR quality of scanned documents sometimes is error-ridden. Most older ETDs were typewritten,

[1]For the rest of the paper, we use *figures* to refer to both *figures* and *tables*.

and figures may have been hand-drawn or rendered in a separate process and literally cut-and-pasted into the typewritten document. In addition, since ETD collections are cross-disciplinary, the documents present a variety of layout and figure styles.

Our work builds on [54], which proposed a novel method for inducing high-quality labels for figure extraction in a large number of scientific documents. Their technique uses data from arXiv and PubMed to locate figures in scientific papers with a reported average precision of 96.8%. Unfortunately, this technique performs worse on scanned documents (as demonstrated later in this document). One of our important contributions in this work is to propose data augmentation methods to improve the performance of identifying figures in scanned ETDs. Our technique takes the data used in the prior work to locate figures in born-digital scientific papers and applies various transformations to the documents in order to mimic older scanned papers.

1.2 Research questions

We explore the possibilities of engineering a system for extracting figures from scanned ETDs. The resultant system should take as input the PDF of the scanned ETD and generate a page-wise list of coordinates for bounding boxes around the figures in each of its pages. To achieve this, we formulated the following research questions:

- **RQ1**: How well can existing methods perform figure extraction from scanned ETDs?

  All of the existing methods for figure extraction have been tested on born-digital ETDs. However, their performance on scanned documents (especially scanned ETDs) has not been explored. Since the scanned and born-digital documents differ consider-

ably in visual appearance, it would be interesting to see how well the existing methods for figure extraction perform on scanned ETDs.

- **RQ2**: Can this performance be improved by using simple data augmentation techniques and weight initialization from the original pre-trained model?

  Since the visual appearance of the scanned and born-digital ETDs differ significantly, it is possible that the performance of the existing methods worsens. In that case, different data augmentation techniques to bring the feature distribution of the born-digital documents closer to that of the scanned documents might create a dataset which can be used to re-train the existing models. It would be interesting to see if the performance improves after re-training.

- **RQ3**: Can this performance be improved by training on manually labelled data?

  The original model as well as the model trained in RQ2 will be trained using data that is automatically labelled. This data was not generated manually. Further, this data is generated or adapted from born-digital sources, which as described above, differ considerably in visual appearance from scanned ETDs. Whether the performance can be improved further using a manually labelled dataset still remains to be explored. This research question aims to address that.

- **RQ4**: Can this performance be improved by using transfer learning techniques?

  All of the previous questions use the pre-trained model for weight initialization, but train all the layers. However, it is often the case that not all the layers of a model need to be trained when its weights are initialized from a pre-trained model from a similar domain. Thus, this research question concerns trying to use transfer learning

to improve the performance by training only a part of the layers.

1.3 Research challenges

1.3.1 PDF file parsing

Although PDF is one of the most preferred formats for working with documents, it is not trivial to parse it. There are tools to convert PDF files into structured formats [13, 16, 17, 35, 42]. However, they are not completely successful. Although transforming a PDF file is a much broader problem, it impacts the ability to extract figures from it. Further, for transforming scanned PDF files into specialized representations, rule-based tools do not work well, since these files are mostly images produced by flat-bed scanners.

1.3.2 Lack of gold standard data

The Deepfigures [54] model uses a dataset labelled using scripts that leverage the LaTeX source of the documents (XML for PubMed). However, for the purpose of figure extraction from scanned scholarly PDF files of ETDs, no such dataset exists. This means that there is a lack of ground truth to validate methods for extracting figures from scanned ETDs.

1.3.3 Big data

In one study [54], the entire arXiv (consisting of LaTeX sources) and PubMed datasets were used to generate labels for figure extraction and training the model. These two datasets consume a significant amount of disk space. Although this work does not use the PubMed dataset, an updated version of the arXiv dataset is used. The total size of the compressed

version of this dataset on disk is about 1.3 TB. Further, when decompressed for processing and converted to its PDF and image formats, it occupies an even larger space. This poses a significant logistical challenge for working with this data and using it in research.

1.3.4 Training deep learning models

Obtaining large amounts of training data, and using that to train models, are key engineering challenges faced by the deep learning community and anyone who deals with deep learning models. Employing a huge amount of data to build large deep learning models with millions of parameters naturally slows down the training process. As the complexity of the model increases, the space-time computational complexity for training and evaluating it increases too. Although we use relatively modern fast CPUs in conjunction with hardware accelerators such as GPUs, the challenge of training the models in a reasonably short amount of time still exists. Various techniques such as training multiple models in parallel and training only a select few parameters of these models have been implemented and described in the latter part of this thesis.

1.4 Research contributions

This thesis, on the topic of figure extraction from scanned ETDs, has the following research contributions:

- **We evaluate the performance of the original Deepfigures model [54] on scanned ETDs.**

  Since the original Deepfigures model [54] was trained on born-digital scholarly documents, we expand the knowledge about its utility by evaluating its performance on

scanned ETDs. We assess it using the F1-score, along with other metrics.

- **We explore whether the original Deepfigures model can be improved for figure extraction from scanned ETDs using data augmentation techniques and weight initialization.**

  Scanned ETDs differ significantly in visual appearance from born-digital ETDs. Since the original Deepfigures model [54] is trained on born-digital documents, we evaluate its performance on a dataset augmented by various techniques which attempt to change the visual appearance of the born-digital documents to look similar to scanned ETDs.

- **We create a new gold standard dataset for figure extraction from scanned ETDs.**

  Because of the lack of a good ground truth dataset for figure extraction from scanned ETDs, there was no way to establish reliable performance metrics of any method that aims to achieve that goal. Hence, we create a gold standard dataset containing more than 10.1k images with more than 3.3k labels for figures.

- **We explore whether using the gold standard dataset for training improves performance.**

  Since the gold standard dataset is labelled manually, it is safe to assume that there is no (or negligible) noise in its labels. Therefore, we train and evaluate our model on the gold standard to explore whether training our model on noise-less data improves the performance.

- **We explore whether transfer learning and weight initialization improve the performance.**

  Since we are already initializing the weights of the model using the pre-trained weights of the original Deepfigures model [54], the convolutional layers retain most of the

features trained for a similar task. Therefore, after weight initialization we freeze the weights of the convolutional layers and only train the final fully connected layers. We evaluate the performance of this model and report our findings.

1.5 Thesis organization

The rest of this thesis is organized as follows:

In Chapter 2, we introduce the related work in a topic-wise fashion in order to offer a broader understanding of the domain.

In Chapter 3, we describe the different data augmentation techniques we used, the overall data generation pipeline from the arXiv LaTeX source files, the process of curating and labelling the gold standard dataset, and other relevant techniques used to prepare the data for experiments.

In Chapter 4, we present the various experiments conducted to answer the proposed research questions. We also present the results from these experiments and discuss the findings.

Finally, in Chapter 6, we conclude this thesis by revisiting research questions and contributions, and suggesting future work.

Chapter 2

Review of Literature

This thesis focuses on extracting figures from scanned ETDs. Many recent techniques have been proposed for extracting figures from born-digital documents. Thus, it is important to understand the past techniques and the relevant background about this field to get a better understanding. In this chapter, we first describe the relevant terms of this field. Then we introduce the past methods of extracting figures, and progressively move to the more recent ones.

2.1 Machine learning literature

2.1.1 Standard computer vision deep learning architecture

The basic neural network is a multi-layer perceptron with fully connected layers. However, for data with larger input dimensions this architecture quickly becomes computationally expensive. Therefore, recent deep learning architectures for computer vision applications first tend to extract higher level features, typically by using deep convolutional neural networks (CNNs) [21, 38, 55]. These features are also known as image embeddings. These embeddings are then passed as input to a final stack of layers, typically fully connected layers. Many standard architectures rely on this approach for computer vision tasks [54] (Figure 2.1).

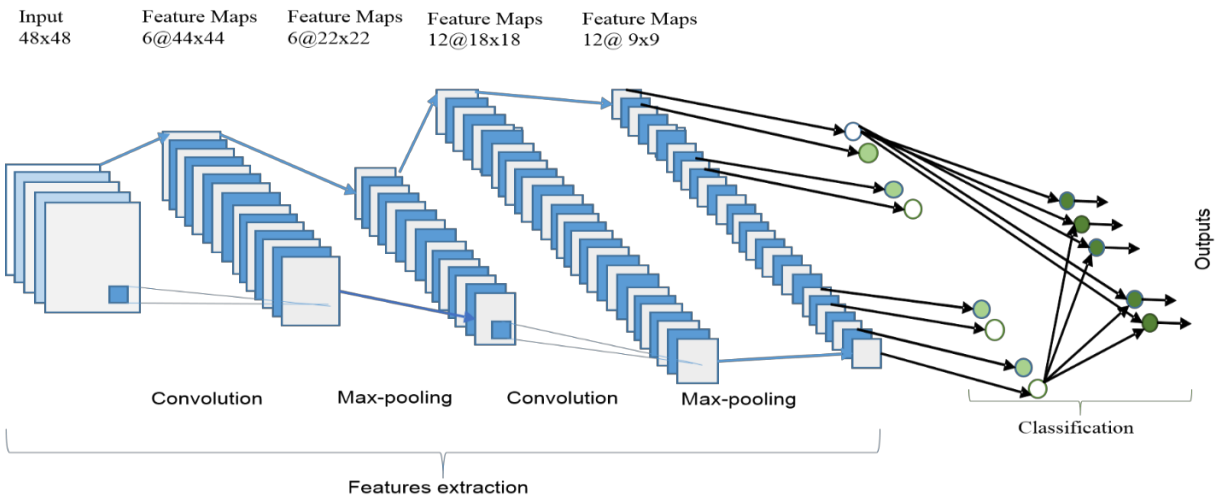


Figure 2.1: This is the overall architecture of several popular CNN models. The initial layers are usually a combination of convolution and max-pooling. Finally, there is a fully connected layer and a classification layer [21].

2.1.2 ResNet-101

ResNets [33], proposed in 2015, have become a popular choice to be used as the initial convolutional layers in a typical CNN architecture. These are especially helpful because of their ease of training while being able to stack multiple layers on top of each other. ResNets use a special type of architecture which uses skip connections between layers (Figure 2.2). Because of these skip connections, the gradients propagating backwards from the layers of the network do not vanish or explode. This enables effective training of the earlier layers of deep neural networks, thereby increasing their learning capacity and improving the overall performance [54].

2.1.3 Overfeat

Overfeat [53] was proposed in 2013 as a new way to predict the bounding boxes of objects in images. Given an image, the task of predicting whether a certain object is present in

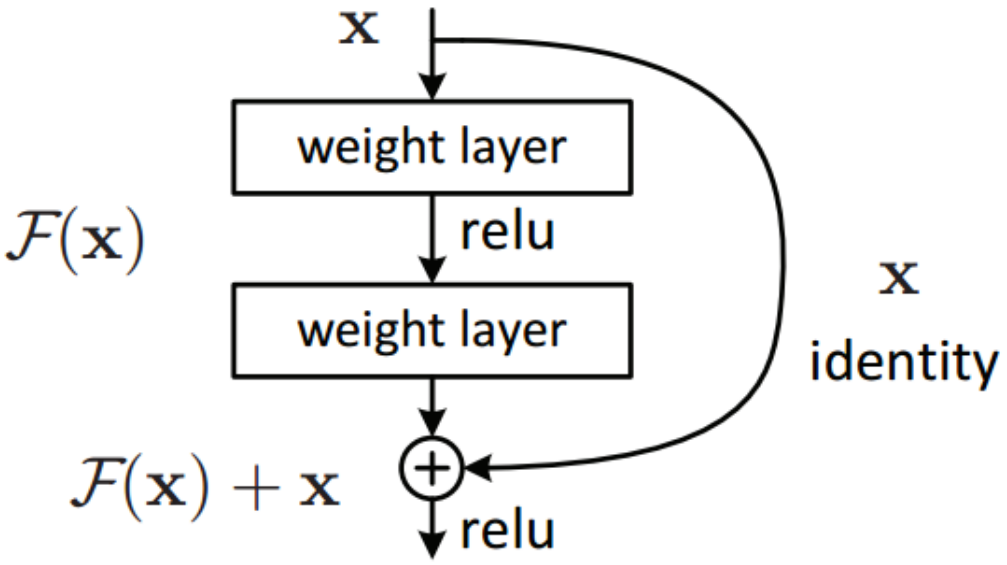


Figure 2.2: A single building block of ResNet showing the skip connection [33].

the image is called classification. However, if the object is present in the image, the task of predicting the location of that object is called object detection. This task is more complex because instead of simply generating a binary output of whether a given object is present in the image or not, the model now needs to predict the coordinates of the bounding box enclosing the object. Further, the presence of multiple objects in a single image makes it even more challenging, because now the dimension of the output to be produced by the model is variable. A quick fix for this is to split the image into parts and run inference on each part. However, this method quickly becomes computationally expensive.

Overfeat uses the latter method described above to run the model fully convolutionally on the input image. The input image is split up using grids with cells of size 20 pixels by 15 pixels. These grids are passed as input to a few convolutional layers and then a couple of fully connected layers to generate the bounding box coordinates as well as the confidence of these coordinates. Finally, the output of all these images is stitched together and non-maximal suppression is applied to get the final result.

2.1.4 YOLO

YOLO is the acronym for You Only Look Once. It is a popular computer vision object detection deep learning model. It is capable of detecting multiple objects in an input image in a single inference pass. Further, it is also well-known for its lower space and time complexity during inference. This makes it a good alternative for deployment on edge devices where low resource consumption is vital.

The first version of YOLO (i.e., YOLOv1) was proposed by Redmon et al. in 2016 [50]. This was the first work which, instead of re-purposing classifiers as object detectors, framed object detection as a regression problem. Since YOLOv1 detects multiple bounding boxes in a single forward pass, it can be trained end-to-end directly for detection performance.

Many subsequent versions of YOLO were proposed by various authors in the following years. In 2020, the 5th version of YOLO (i.e., YOLOv5) was proposed. This model was released by Ultralytics LLC. on Github [12]. YOLOv5 achieves higher detection performance than all of the previous versions. Further, YOLOv5 comes in 4 different network sizes (small, medium, large, and extra large). These various network sizes allows users to make a trade-off between the time and space complexity. The exact details about the number of parameters and the run times for each size variant of YOLOv5 are documented [12]. In our experiments, we use the extra large version of YOLOv5 which has about 89 million trainable parameters.

Table 2.1 compares the Deepfigures and YOLOv5 model across the number of trainable parameters in the model, the inference times averaged across 500 images that were run on an Nvidia Tesla P100 GPU, and the year in which the model was published. The inference times in Table 2.1 do not include any time needed for pre-processing the image and post-processing the predictions. We measure only the time to make a single forward pass for a single pre-processed image on the GPU.

Table 2.1: Comparison of the features of the Deepfigures model and YOLOv5.

Model	No. of params.	Inference time	Year published
Deepfigures	45 million	33.514 ms	2018
YOLOv5 (extra large)	89 million	35.048 ms	2020

We can observe that the inference times are almost the same even though the number of parameters in YOLOv5 is almost double than that of Deepfigures. Our intuition says that the time required to transfer data to and fro between the GPU and CPU could be a major chunk of the inference times.

2.1.5 Weight initialization

Weight initialization in a deep learning model is the process of assigning values to the weights and biases of each of its layers. Weight initialization is crucial for effective training of the model and can make a huge impact on the convergence of the model and the time it takes to achieve that.

One of the earliest methods for weight initialization is to do so randomly. It is common to initialize all of the weights to close to zero, with just an epsilon of random deviation from zero. This helps to break the initial stagnancy of the model while keeping it unbiased.

Another common technique was proposed by He et al. in their 2015 paper [32]. In this technique, the bias vectors are initialized to zero. The weights are first chosen randomly from a standard normal distribution. Each of these randomly chosen weights is multiplied by $\sqrt{2/n}$, where n is the total number of incoming connections into the current layer from the output of the previous layer.

However, given a pre-trained model, it is common to use its weights as the initial weights for further training the model on a similar but different dataset. This brings us to the concept

of transfer learning.

2.1.6 Transfer learning

Using the weights of a pre-trained model as the initial weights for training on another, requires both the models to have the same or a similar architecture. This, as discussed before, is a type of weight initialization. However, another common technique when given a pre-trained model and a similar task is to train a part of the model. In other words, the new model is initialized with the weights of the pre-trained model. Further, the weights of some of the layers are kept unchanged while training the rest of the model on the new dataset. This technique assumes that the high-level features learnt by the frozen layers can be safely transferred to the new task. This often requires a lesser amount of data since not all layers are training. This is also less computationally expensive. This technique is known as fine-tuning a pre-trained model for a new task, and is a part of transfer learning [49, 57].

2.1.7 Data labelling tools

Machine learning models often rely on labelled data for their training. Further, deep learning models require huge amounts of data because the features in the data useful for predictions are learnt automatically. Whenever labelled data is not available, ML researchers often create the labels manually (either through crowdsourcing or themselves). For object detection tasks, many tools have been developed and made open source to effectively create these labels.

VGG Image Annotator (VIA) [27, 28] is one such tool for manually creating bounding box labels on images. This is a light-weight tool which can be run entirely in an Internet browser. It is written using HTML and JavaScript. It has the capability to export annotations in CSV as well as the popular COCO format 3.10.

2.1.8 Dataset formats

With the growth of research in the field of machine learning, researchers started to come up with a variety of datasets for different tasks. However, for some common tasks, like object detection, image classification, etc., the input format of data is often the same. This means that with a unified format of dataset, the software needed to read it can be reused. Therefore, multiple new dataset formats have sprung up, often named after a popular dataset in the domain. A few examples of these formats are the COCO object detection format [3], Pascal VOC [8], CIFAR-10, CIFAR-100 [2], and so on. These popular datasets are often supported by prominent machine learning libraries such as TensorFlow and PyTorch with packaged code to read and load them [9, 10]. Therefore, any dataset in these popular formats becomes easier to load by re-using the code in these machine learning libraries.

2.1.9 Common metrics in machine learning for object detection

During training any machine learning model, it is important to measure how accurate it is. This gives us a better sense of how well the model is training. For binary classification tasks, a common metric is to measure the accuracy. However, accuracy is not a reliable metric, especially when the training set labels are imbalanced. Therefore, precision, recall and F1-score are some other metrics used in conjunction with accuracy. These are calculated as follows:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

For object detection tasks, computing these metrics is different because predicting a bounding box is not a binary classification task. Therefore, the intersection over union (IOU, as in Figure 2.3) is computed between the predicted and true bounding boxes. A predicted box is considered a true positive if its IOU is over a certain predefined threshold, but a false positive otherwise [37].

2.1.10 Data augmentation

Data augmentation [48] is a popular technique in deep learning which helps to train a model better without collecting any new data. With this technique, the existing training data is modified and the deep learning model is re-trained on the modified data. Tanner et al. [56] list the techniques frequently used for computer vision tasks, such as affine transformations, random rotations, additive noise (salt-and-pepper, Gaussian, etc.), perspective transformations, random cropping, and so on. A number of open-source implementations have emerged online for implementing these techniques. ImgAug [34] is one of the popular tools which we use in this project. It provides the capability to augment not only the image, but also the corresponding bounding boxes in the image. For example, if a certain degree of affine

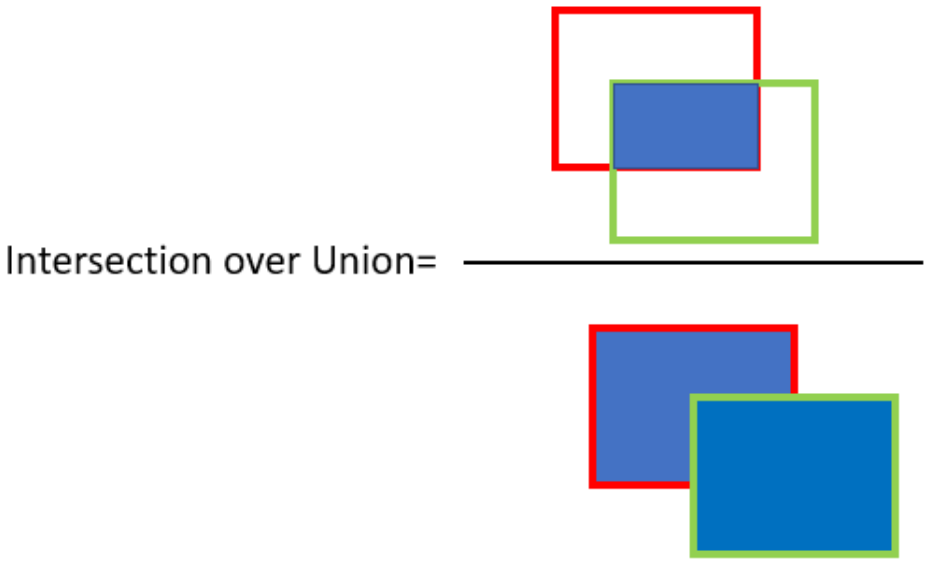


Figure 2.3: Computing the Intersection Over Union [37].

transformation is applied on an image, the corresponding bounding boxes will be adjusted in order to accommodate for this transformation.

2.2 Older methods for PDF file parsing

In the following subsections we describe the various methods used in the past to parse PDF files. Numerous open-source tools are available on the Internet which are able to parse and extract some parts of the PDF file.

2.2.1 PDFEdit

PDFEdit [35] is an open-source library which is based on the XPDF [19] toolkit. It also has a GUI version available. With PDFEdit, it is possible to look into the underlying object tree of a given PDF file. However, parsing images from these object streams is not trivial since they are often represented in non-standard formats.

2.2.2 PDFAct

PDFAct [16] is another open-source tool which is built to extract structure from PDF files of scientific articles. It takes in a PDF file as input and outputs a text file containing the entire text in the PDF file. However, this tool does not extract the images.

2.2.3 PDFToCairo

Another interesting tool is PDFToCairo [17]. This tool converts PDF files using the Cairo output device [14] of the poppler PDF library [18] to either an image format or the PostScript (PS) format. Using PDFToCairo, it is possible to convert a PDF file to a final .eps format on a Unix system by converting it to an intermediate format and filtering out some elements from it. The resultant filtered .eps file does not contain any text. A sample command [30] to achieve this is as follows:

```
pdftocairo -f 4 -l 4 -eps input.pdf  - | sed '/^BT$/,/^ET$/ d' > output.eps
```

When Figures 2.4 and 2.6 were processed using this command, the outputs extracted are shown in Figures 2.5 and 2.7. In Figure 2.7, it can be seen that not all the components of

the figures have been extracted. This shows that this tool only extracts the vector graphics from PDF files. Further, this tool works only on born-digital PDF files.

2.2.4 APDFL

APDFL [13] is another library, developed by Datalogics Inc. for extracting the various components of a PDF file. However, similar to PDFToCairo, this library can't extract all types of figures from born-digital PDF files nor extract any figures from scanned PDF files.

2.3 Newer methods for PDF parsing

2.3.1 Grobid

Grobid [42] was started as a hobby in 2008 and was made open-source in 2011. It is a machine learning library for extracting, parsing, and re-structuring PDF documents into TEI/XML format. Grobid especially focuses on technical and scientific documents. However, the output of Grobid is only the text of the PDF file. It cannot extract any graphical elements (e.g., figures) of the PDF document.

2.3.2 PDFFigures2

Clarke and Divvala [26] proposed a new approach which analyzes the structure of individual pages by detecting chunks of body text, graphical elements, and captions, and then locates figures by reasoning about the empty regions within that text. Their results are used as a baseline for their future work – Deepfigures. However, similar to other tools described above, PDFFigures2 relies on a rule-based system to extract or parse PDF files. Therefore, it only

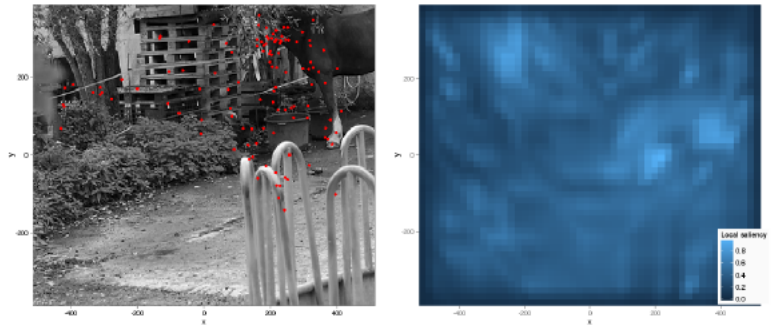


Figure 3: An image from the dataset of Kienzle et al. (2009), along with an "interest map" - local saliency computed according to the Itti-Koch model (Itti and Koch, 2001; Walther and Koch, 2006). Fixations made by the subjects are overlaid in red. How well does the interest map characterise this fixation pattern? This question is not easily answered by eye, but may be given a more precise meaning in the context of spatial processes.

3.1 Understanding the role of covariates in determining fixated locations

To be able to move beyond the basic statement that local image cues somehow *correlate* with fixation locations, it is important that we clarify how covariates could enter into the latent intensity function. There are many different ways in which this could happen, with important consequences for the modelling. Our approach is to build a model gradually, starting from simplistic assumptions and introducing complexity as needed.

To begin with we imagine that local contrast is the only cue that matters. A very unrealistic but drastically simple model assumes that the more contrast there is in a region, the more subjects' attention will be attracted to it. In our framework we could specify this model as:

$$\eta(x, y) = \beta_0 + \beta_1 c(x, y)$$

However, surely other things besides contrast matters - what about average luminance, for example? Couldn't brighter regions attract gaze?

This would lead us to expand our model to include luminance as another spatial covariate, so that the log-intensity function becomes:

$$\eta(x, y) = \beta_0 + \beta_1 c(x, y) + \beta_2 l(x, y)$$

in which $l(x, y)$ stands for local luminance. But perhaps edges matter, so why not include another covariate corresponding to the output of a local edge detector $e(x, y)$? This results in:

$$\eta(x, y) = \beta_0 + \beta_1 c(x, y) + \beta_2 l(x, y) + \beta_3 e(x, y)$$

It is possible to go further down this path, and add as many covariates as one sees fit (although with too many covariates, problems of variable selection do arise, see Hastie et al., 2003), but to make our lives simpler we can also rely on some prior work in the area and use pre-existing, off-the-shelf *image-based saliency models* (Fecteau and Munoz, 2006). Such models combine many local cues into one interest map, which saves us from having to choose a set of covariates and then estimating their relative weight (although see Vincent et al., 2009 for work in a related direction). Here we focus on the perhaps most well-known among these models, described in Itti and Koch (2001) and Walther and Koch (2006), although many other interesting options are available (e.g., Bruce and Tsotsos, 2009, Zhao and Koch, 2011, or Kienzle et al., 2009).

7

Figure 2.4: Input to PDFToCairo

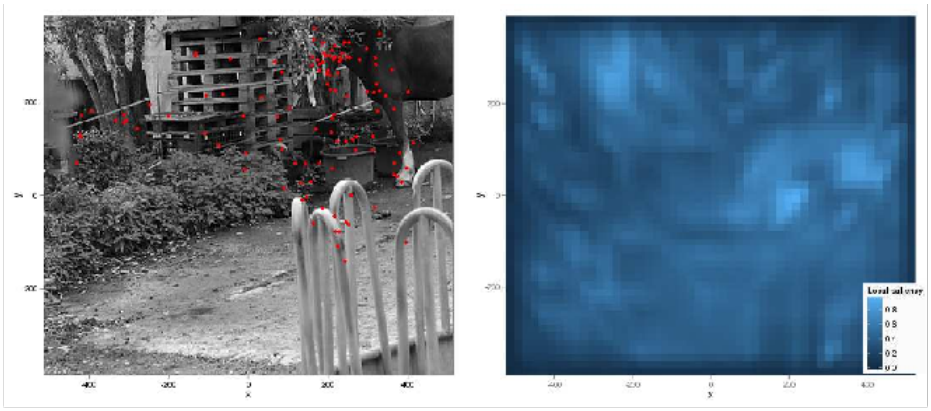


Figure 2.5: Output from PDFToCairo showing the successfully extracted figures

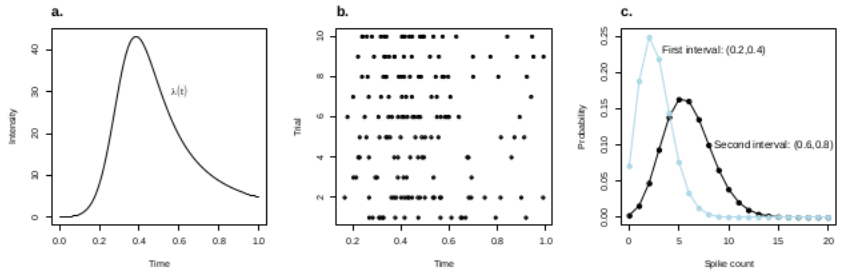


Figure 1: A first example of a point process: the Inhomogeneous Poisson Process (IPP) as a model for spike trains. **a.** The neuron is assumed to respond to stimulation at a varying rate over time. The latent rate is described by an intensity function, $\lambda(t)$ **b.** Spikes are stochastic: here we simulated spike trains from an IPP with intensity $\lambda(t)$. Different trials correspond to different realisations. Note that a given spike train can be seen simply as a set of points in $(0, 1)$. **c.** The defining property of the IPP is that spike counts in a given interval follow a Poisson distribution. Here we show the probability of observing a certain number of spikes in two different time intervals.

2.1 Definition and examples

In statistics, point patterns in space are usually described in terms of point processes, which represent realisations from probability distributions over sets of points. Just like linear regression models, point processes have a deterministic and a stochastic component. In linear models, the deterministic component describes the average value of the dependent variable as a function of the independent ones, and the stochastic component captures the fact that the model cannot predict perfectly the value of the independent variable, for example because of measurement noise. In the same way, point processes will have a latent *intensity function*, which describes the expected number of points that will be found in a certain area, and a stochastic part which captures prediction error and/or intrinsic variability.

We focus on a certain class of point process models known as inhomogeneous Poisson processes. Some specific examples of inhomogeneous Poisson processes should be familiar to most readers. These are temporal rather than spatial, which means they generate random point sets in time rather than in space, but equivalent concepts apply in both cases.

In neuroscience, Poisson processes are often used to characterize neuronal spike trains (see e.g., Dayan and Abbott, 2001). The assumption is that the number of spikes produced by a neuron in a given time interval follows a Poisson distribution: for example, repeated presentation of the same visual stimulus will produce a variable number of spikes, but the variability will be well captured by a Poisson distribution. Different stimuli will produce different average spike rates, but spike rate will also vary over *time* during the course of a presentation, for example rising fast at stimulation onset and then decaying. A useful description, summarized in Figure 1, is in terms of a latent intensity function $\lambda(t)$ governing the expected number of spikes observed in a certain time window. Formally, $\int_{\tau}^{\tau+\delta} \lambda(t)\,\mathrm{d}t$ gives the expected number of spikes between times τ and $\tau + \delta$. If we note $\mathbf{t} = t_1, t_2, \ldots, t_k$ the times at which spikes occurred on a given trial, then $\mathbf{t}$ follows a inhomogeneous Poisson Process (from now on IPP) distribution if, for all intervals $(\tau, \tau + \delta)$, the number of spikes occurring in the interval follows a Poisson distribution (with mean given by the integral of $\lambda\,(t)$ over the interval).

The temporal IPP therefore gives us a distribution over sets of points in time (in Figure 1, over the interval $[0, 1]$). Extending to the spatial case is straightforward: we simply define a new intensity function $\lambda(x, y)$ over space, and the IPP now generates point sets such that the expected number of points to appear in a certain area A is $\int_A \lambda\,(x, y)\,\mathrm{d}x\mathrm{d}y$, with the actual quantity again following a Poisson distribution. The spatial IPP is illustrated on Figure 2.

4

Figure 2.6: Another input to PDFToCairo

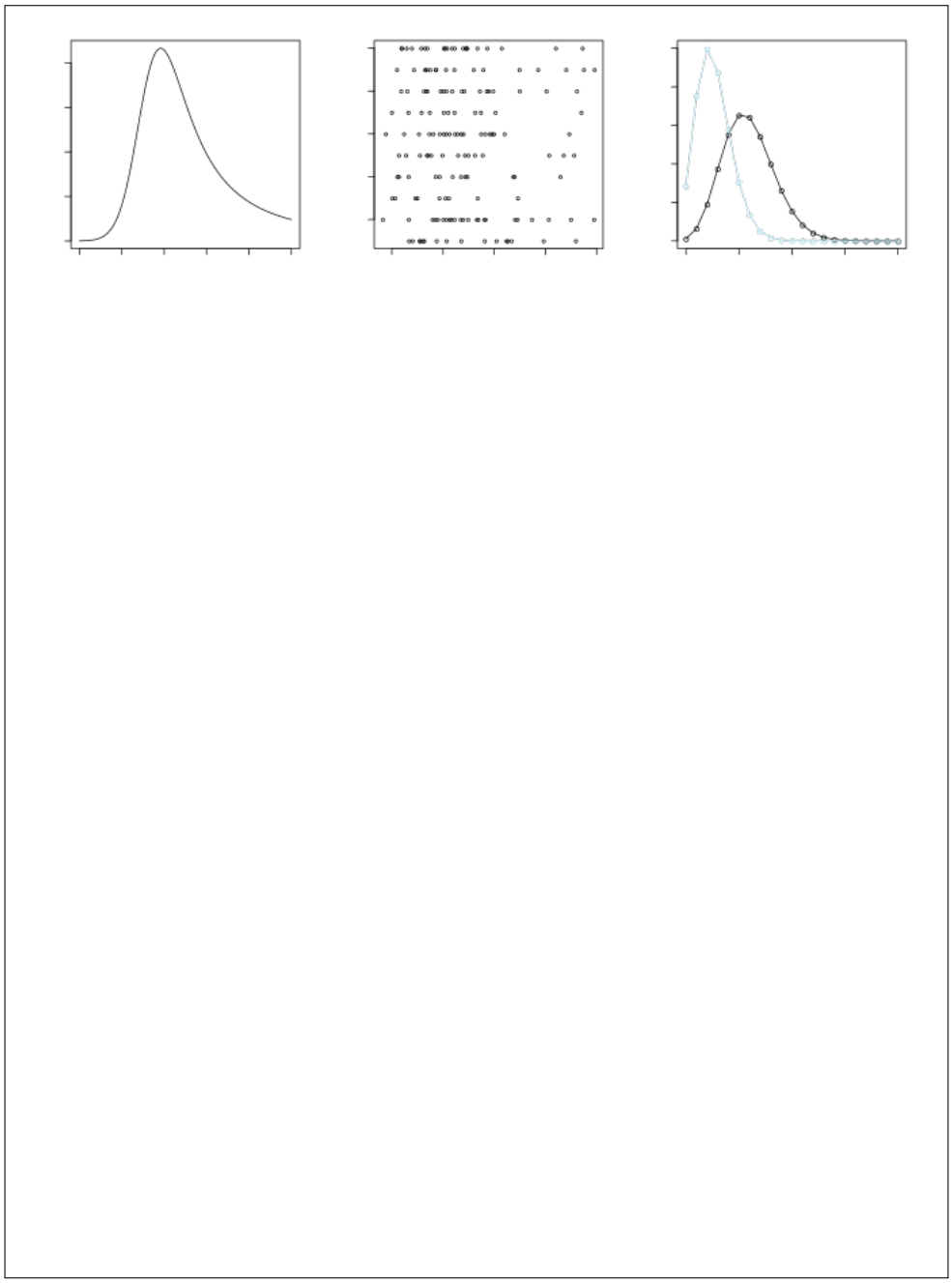


Figure 2.7: Output from PDFToCairo showing the partially extracted figures

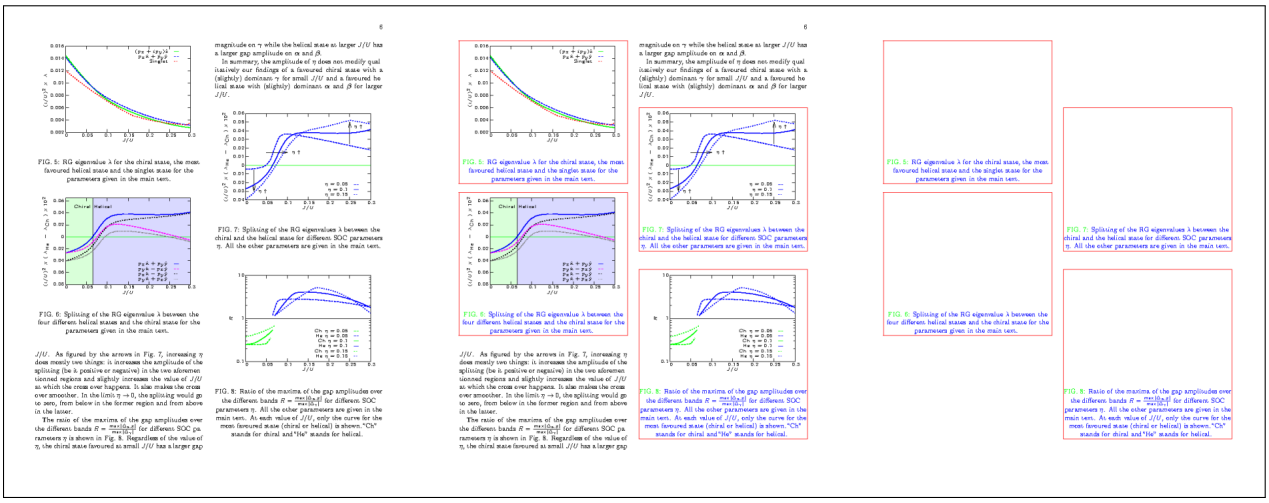


Figure 2.8: Sample images in the Deepfigures pipeline [54]. The figure to the left is the original image of the page of the PDF file. The figure in the middle is the image of the page with bounding boxes drawn around its figures. The figure to the right is the result of pixel-by-pixel subtraction of the first two figures.

works for born-digital PDF files and not for scanned ones.

2.3.3 Deepfigures

Siegel et al. proposed Deepfigures [54], a method for extracting non-textual components (charts, figures, and tables) from scholarly PDF files. Data from arXiv [58] and PubMed [24] is used to locate figures in scientific papers. For the arXiv dataset, the authors modify the original LaTeX source code of the PDF files such that it renders visible bounding boxes around figures and captions. Figures are already present in a parsed format of the PubMed dataset. These induced labels were then used to train a deep learning model (ResNet-101 [33] in conjunction with the Overfeat architecture [53]) to predict the coordinates of the bounding boxes around figures. The number of trainable parameters in the Deepfigures model total about 45 million. The figures were then extracted by cropping these predicted bounding boxes out from the rendered PDF files. (Figure 2.8).

2.3.4 TableBank

Li et al. [41] proposed TableBank, an image-based table detection and recognition program and dataset built on top of Deepfigures. Their contribution is an ML model with weak supervision on a dataset of Word and LaTeX documents crawled from the Internet. The authors claim that they do not filter the language or the domain of the documents being crawled, which makes their dataset more diverse and robust. As with Deepfigures, this technique involves modifying the source code of the document to introduce a bounding box around figures. Li et al. take the idea a step further by building an RNN model for converting detected tables (which are in image format) into a table markup format (i.e., a table parsed into text). In other words, it converts an image of a table into a structured table using RNNs. They call this process table structure recognition.

2.3.5 Faster R-CNN for table extraction

More recently, Hansen et al. [31] accomplished marginal improvements on the performance of Deepfigures by employing an objection detection model, Faster R-CNN, which allowed them to achieve better region assignments for tables in a PDF document. Further, they also introduced a dataset with 31,639 manually labeled pages of PDF files with image bounding boxes.

2.3.6 Faster R-CNN for figure extraction

In May 2020, Lee et al. proposed the Newspaper Navigator dataset as a part of their paper [40]. They created this dataset through crowdsourcing where each user added annotations of 7 categories (headlines, photographs, illustrations, maps, comics, editorial cartoons, and

advertisements) to a corpus of scanned historic newspapers from Chronicling America. More than 48 thousand annotations were created across these 7 categories. These annotations were validated through multiple users. Further, Lee et al. fine-tuned a pre-trained Faster R-CNN to transfer its learning for detecting bounding boxes by training on this dataset. Inference was then run on 16 million historical newspaper pages from Chronicling America. The work in this paper comes closest to extracting figures from scanned ETDs.

All the techniques for parsing PDF files discussed till now work only on born-digital documents or are not demonstrated to work on scanned ETDs.

2.4 Born-digital vs. scanned ETDs

Before the late twentieth century, ETDs were either hand-written or typed using a typewriter on paper (e.g., [23, 25]). However, as personal computers gradually became ubiquitous, people started typing ETDs using software such as word processors and more recently LaTeX, and were stored in digital libraries usually in PDF (e.g., [22, 54]). The former type of ETDs were later digitized using scanners into PDF and then stored in digital libraries. As a result, today's digital libraries typically consist of two types of ETDs - scanned, and born-digital - representing the two types of ETDs described above. This effect is not only limited to ETDs, but also to any general scholarly documents, or sometimes general PDF files as well.

Chapter 3

Data generation pipeline and pre-processing

Since our work builds upon Deepfigures [54], in this chapter we first review the general strategy employed by Deepfigures [54]. Then we describe our modifications to this pipeline, followed by our system to run this pipeline at scale.

3.1 Approach of Deepfigures [54]

The overall approach is to generate high quality labels for figure extraction on a large number of scientific documents. The generated dataset is then used to train a deep learning model which is a combination of ResNet-101 and the Overfeat architecture.

3.1.1 Label induction

This phase of Deepfigures deals with two datasets — arXiv and PubMed.

arXiv dataset

First is the arXiv[1] dataset which can be obtained using AWS's S3 API[2] as mentioned on their bulk access website [58].

Modifying the LaTeX source: This dataset contains the LaTeX source code for each research paper in the dataset. This LaTeX source code is first used to compile the PDF file which is then converted to a list of images for each of its pages. Further, the LaTeX source code is modified to add bounding boxes around each figure. This is achieved by adding a few lines of LaTeX code at the beginning of the LaTeX source code.

Specifically, the following changes are made to the LaTeX source files:

```
\usepackage{color}
\usepackage{floatrow}
\usepackage{tcolorbox}

\DeclareColorBox{figurecolorbox}{\fcolorbox{red}{white}}
\DeclareColorBox{tablecolorbox}{\fcolorbox{yellow}{white}}

\floatsetup[figure]{framestyle=colorbox,
    colorframeset=figurecolorbox,framearound=all,
    frameset={\fboxrule1pt\fboxsep0pt}}
\floatsetup[table]{framestyle=colorbox,
    colorframeset=tablecolorbox,framearound=all,
    frameset={\fboxrule1pt\fboxsep0pt}}
```

[1] https://arXiv.org
[2] https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html

The first three lines declare the packages to add the bounding boxes around figures. The next two lines declare the colours of the borders of the bounding boxes. Finally, the last two lines use these declarations to modify the rendering options of all figures so that a border of 1 pixel thickness is drawn around all the figures in the entire document. These modified LaTeX files are similarly compiled into PDF files and converted to a list of images for each of its pages.

Generating the labels: Now, for each scholarly paper in the dataset, we are left with two lists of images - the first does not contain any bounding boxes around its figures while the second does. Each image in these two lists represents a single page from the PDF document. Each corresponding pair of images from these two lists are subtracted in a pixel-by-pixel fashion to obtain a list of images with only the bounding boxes around the figures. For example, the image for the n-th page from the first list is subtracted from the image for the n-th page from the second list. The connected components in the resultant images are used to find the coordinates of the bounding boxes.

PubMed dataset

Labels are also generated for the PubMed dataset. However, the PubMed dataset, along with the paper PDF files, contains auxiliary data for better user experience. This auxiliary data includes markup indicating the location of the captions and the image files for all figures. These image files are used to do a multi-scale template matching on the page images of the original paper to obtain the location of the figures in the page. The locations are then used as labels for training the Deepfigures model.

Evaluating label quality and training

The quality of these labels is evaluated by manually labelling a few randomly sampled images and comparing these against the generated labels. These bounding boxes are then used to train a deep learning object detection model (ResNet-101 [33] + Overfeat [53]) (Figure 3.1).

3.2 Feature distribution of datasets

3.2.1 Feature distribution of born-digital PDF files

Part of our work is based on the data generation pipeline in Deepfigures [54], reviewed in Section 3.1. In Deepfigures, the labelled data is generated from two datasets, i.e., arXiv and PubMed. In the case of arXiv, the LaTeX source of each paper is used to generate the labels. This means that the compiled PDF file is 'born-digital' and not scanned (for example using a flat-bed scanner) (Figure 3.2). Similarly, for the PubMed dataset, most of the PDF files used to generate the labels are born-digital. Further, for the older PDF files, although they are scanned, the figure/caption markup in the auxiliary data is not reliable, sometimes non-existent. As can be seen in example documents [23, 29, 44, 52], although we can see the figures in the PDF file, they are not always present in the markup. This implies the presence of noise in the PubMed training data.

3.2.2 Feature distribution of scanned PDF files

As opposed to born-digital PDF files, scanned PDF files, at some point of time, existed as a hard-copy and were later digitized into PDF using scanning tools such as flat-bed scanners (Figure 3.3). The process of scanning introduces certain artifacts in the PDF file. For

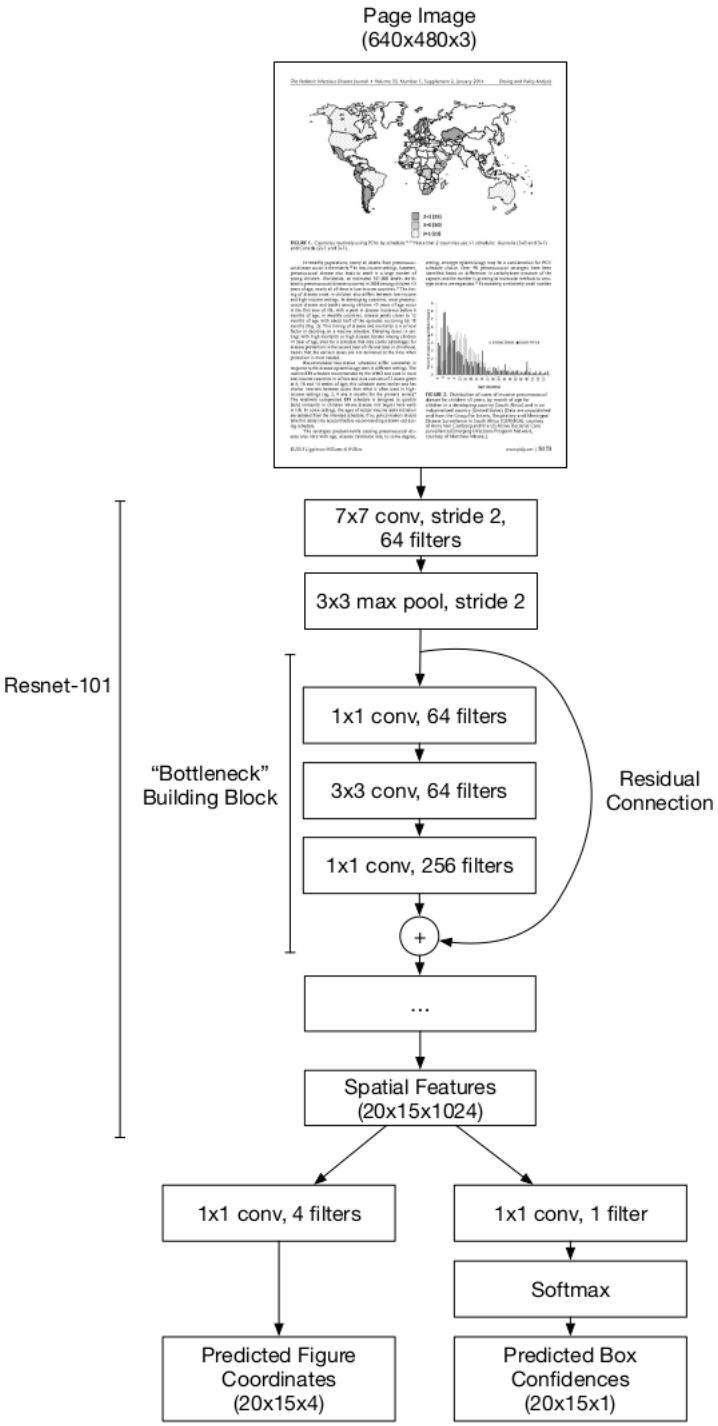


Figure 3.1: The architecture of the deep learning model used in Deepfigures [54]. The upper part is ResNet-101 and the lower part is the Overfeat network for bounding box regression.

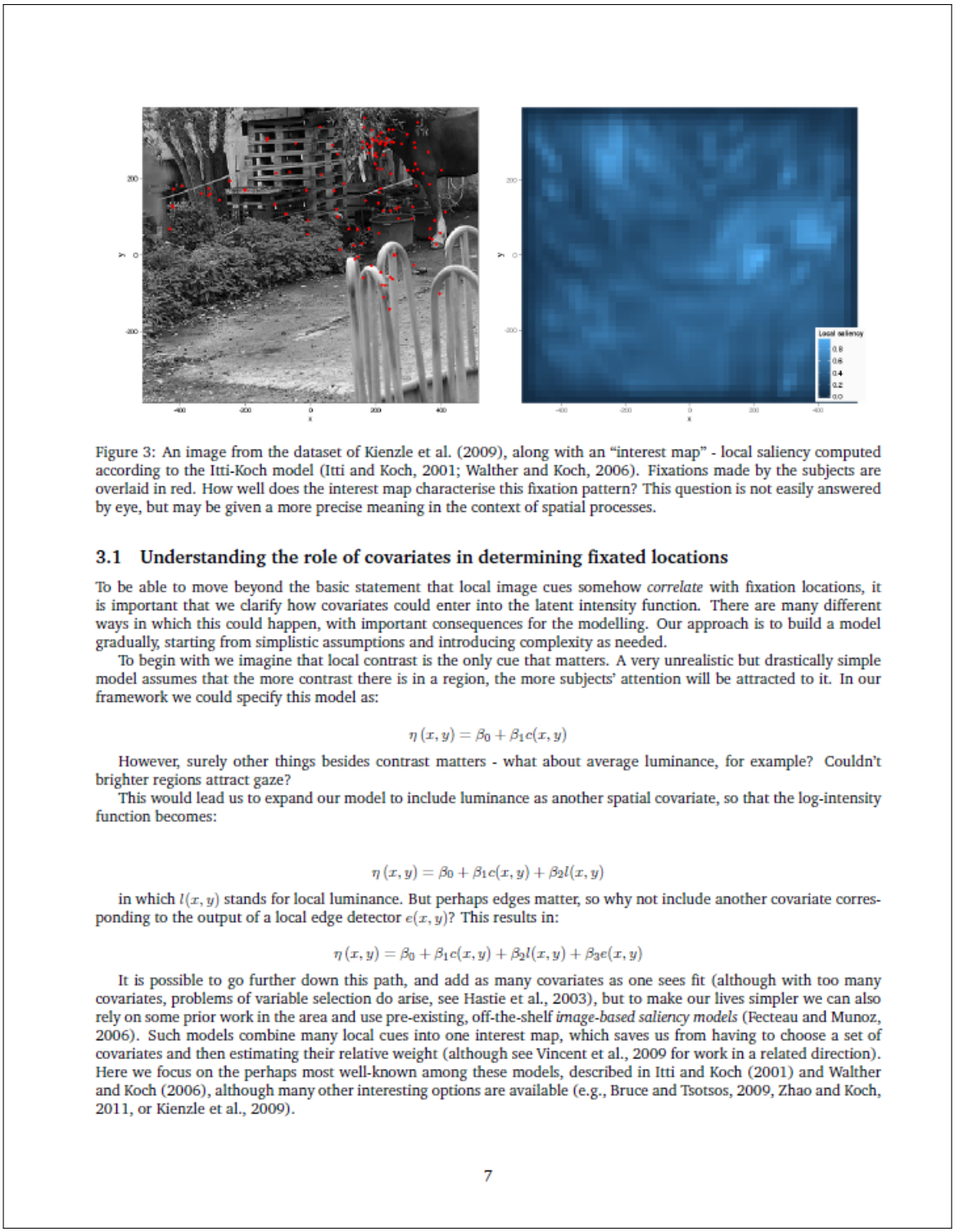


Figure 3: An image from the dataset of Kienzle et al. (2009), along with an "interest map" - local saliency computed according to the Itti-Koch model (Itti and Koch, 2001; Walther and Koch, 2006). Fixations made by the subjects are overlaid in red. How well does the interest map characterise this fixation pattern? This question is not easily answered by eye, but may be given a more precise meaning in the context of spatial processes.

3.1 Understanding the role of covariates in determining fixated locations

To be able to move beyond the basic statement that local image cues somehow *correlate* with fixation locations, it is important that we clarify how covariates could enter into the latent intensity function. There are many different ways in which this could happen, with important consequences for the modelling. Our approach is to build a model gradually, starting from simplistic assumptions and introducing complexity as needed.

To begin with we imagine that local contrast is the only cue that matters. A very unrealistic but drastically simple model assumes that the more contrast there is in a region, the more subjects' attention will be attracted to it. In our framework we could specify this model as:

$$\eta(x, y) = \beta_0 + \beta_1 c(x, y)$$

However, surely other things besides contrast matters - what about average luminance, for example? Couldn't brighter regions attract gaze?

This would lead us to expand our model to include luminance as another spatial covariate, so that the log-intensity function becomes:

$$\eta(x, y) = \beta_0 + \beta_1 c(x, y) + \beta_2 l(x, y)$$

in which $l(x, y)$ stands for local luminance. But perhaps edges matter, so why not include another covariate corresponding to the output of a local edge detector $e(x, y)$? This results in:

$$\eta(x, y) = \beta_0 + \beta_1 c(x, y) + \beta_2 l(x, y) + \beta_3 e(x, y)$$

It is possible to go further down this path, and add as many covariates as one sees fit (although with too many covariates, problems of variable selection do arise, see Hastie et al., 2003), but to make our lives simpler we can also rely on some prior work in the area and use pre-existing, off-the-shelf *image-based saliency models* (Fecteau and Munoz, 2006). Such models combine many local cues into one interest map, which saves us from having to choose a set of covariates and then estimating their relative weight (although see Vincent et al., 2009 for work in a related direction). Here we focus on the perhaps most well-known among these models, described in Itti and Koch (2001) and Walther and Koch (2006), although many other interesting options are available (e.g., Bruce and Tsotsos, 2009, Zhao and Koch, 2011, or Kienzle et al., 2009).

7

Figure 3.2: A sample page rendered as an image from a born-digital ETD [22] from the arXiv dataset.

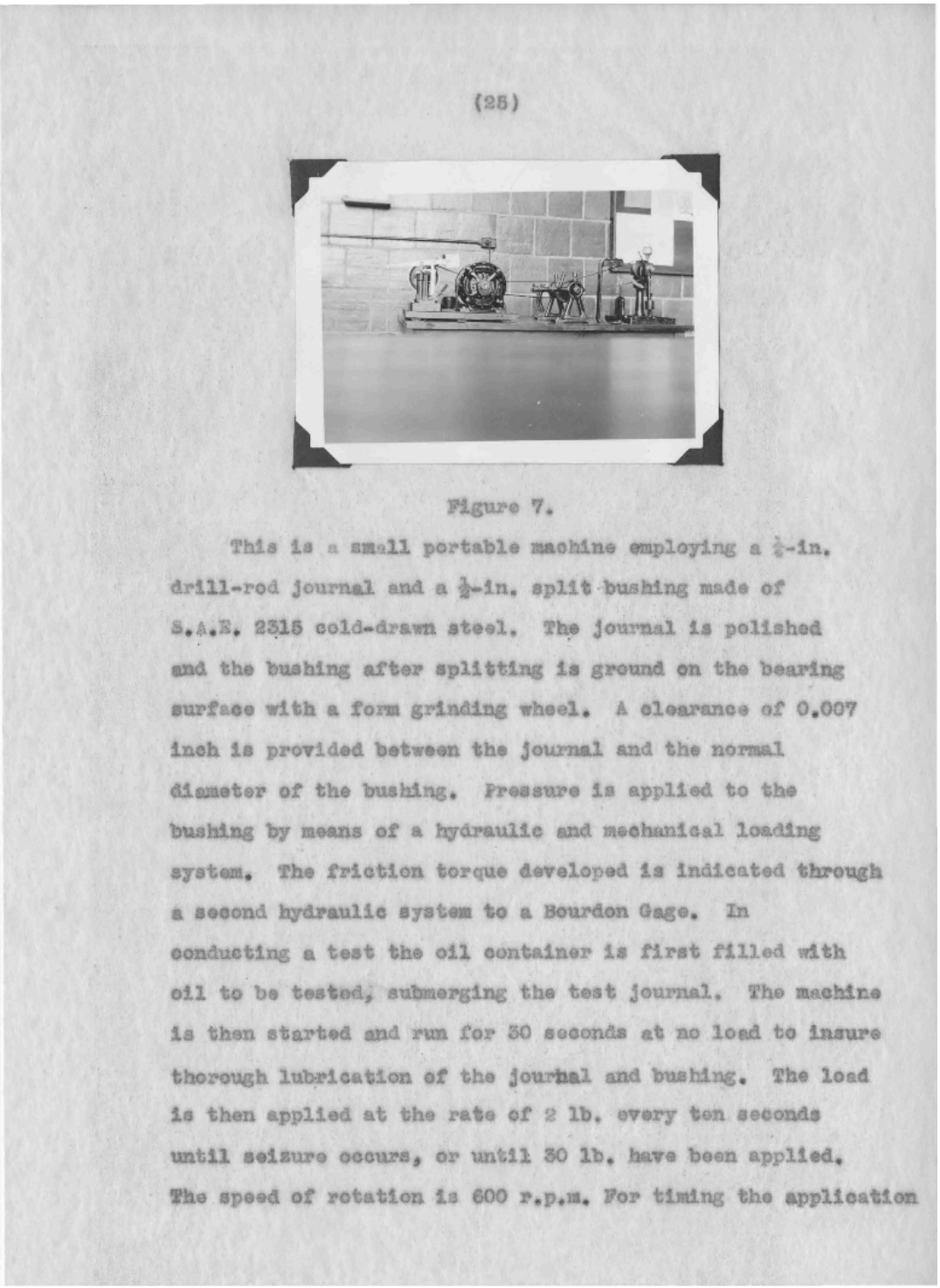
example, the content for some pages might be slightly rotated/tilted because of errors in the placement of paper in the scanner. Other kinds of noise are possible, such as salt-and-pepper noise, blurring, perspective transformations, etc. Further, the content of the PDF files could have been prepared with a typewriter, or hand-written. Therefore, the overall appearance of a scanned PDF file can vary drastically from a born-digital one. As a result, the feature distribution of the training data on which Deepfigures was trained could be significantly different than that of scanned PDF files, thereby resulting in lower performance of Deepfigures on scanned documents. Quantifiable results to support this argument are presented later in this thesis.

3.3 Our modifications to the Deepfigures [54] pipeline

A part of our work builds on the efforts of Deepfigures [54] by using various data augmentation techniques to augment the training data in an attempt to improve performance on older and scanned PDF documents. To the best of our knowledge, no such study was done before.

3.3.1 Bringing the two feature distributions closer

We propose to modify the label generation pipeline of Deepfigures by augmenting the training data used for Deepfigures using various data augmentation strategies. The purpose of this data augmentation process is to bring the feature distribution of the original training data of Deepfigures closer to the feature distribution of scanned documents. The hypothesis behind this is that since the original Deepfigures model was not trained on scanned documents, it will perform worse on them. Therefore, training it on a dataset whose feature distribution is



(25)

Figure 7.

This is a small portable machine employing a ½-in. drill-rod journal and a ½-in. split bushing made of S.A.E. 2315 cold-drawn steel. The journal is polished and the bushing after splitting is ground on the bearing surface with a form grinding wheel. A clearance of 0.007 inch is provided between the journal and the normal diameter of the bushing. Pressure is applied to the bushing by means of a hydraulic and mechanical loading system. The friction torque developed is indicated through a second hydraulic system to a Bourdon Gage. In conducting a test the oil container is first filled with oil to be tested, submerging the test journal. The machine is then started and run for 30 seconds at no load to insure thorough lubrication of the journal and bushing. The load is then applied at the rate of 2 lb. every ten seconds until seizure occurs, or until 30 lb. have been applied. The speed of rotation is 600 r.p.m. For timing the application

Figure 3.3: A sample page rendered as an image from a scanned ETD [25] from VTechWorks.

closer to that of scanned documents should help improve its performance on scanned dataset.

3.3.2 Why just image-based transformations are not enough

The proposed data augmentation can be achieved by modifying the original training data such that its style (visual appearance) resembles that of scanned documents. Since the only input to the deep learning model is the image of each page of the input PDF file, the main aim is to make these input images look visually similar to an image of a scanned document. The obvious choice is to somehow transform the image accordingly. However, we note that simply applying image-based transformations won't change the intrinsic layout of the page. For instance, in Figure 3.2, irrespective of the amount of image-based transformations we apply, features such as the layout of the page, relative positioning of the elements, font type and size, etc. won't change much.

3.3.3 Approach of transformations

To tackle this problem, we propose a two-stepped approach. We leverage techniques similar to the ones employed by Deepfigures. We use a combination of LaTeX-based and image-based transformations. We first alter the LaTeX source code of the documents. We then apply image-based transformations on the resultant document. Following is the description of these two approaches in more detail. It should be noted that these transformations are selected heuristically.

3.3.4 LaTeX-based transformations

We propose the following LaTeX-based transformations:

12pt font

Scanned ETDs were usually written using typewriters. As a result, the font of these documents is often larger than born-digital ones. Usually, in a LaTeX document, the font size is declared in a line that looks like the following:

```
\documentclass[sigconf]{acmart}
```

We increase this font size to a 12pt font by replacing the above line with the following:

```
\documentclass[sigconf,12pt]{acmart}
```

This changes the text font to a 12pt font in the entire document. To find this line in the LaTeX source code, we use the following Python regex:

```
r'\\documentclass\[(.*?)\]\{(.*?)\}'
```

Typewriter font

As mentioned above, since many scanned ETDs are written using typewriters, the visual appearance of their font differs significantly from the font of a born-digital ETD.

For example, the font of a document typed using a typewriter looks like the font seen in Figure 3.3. This font is different from that seen in Figure 3.2.

Another important characteristic of a document prepared with a typewriter is that not all the characters in the text are of equal brightness/darkness. This can be seen in the third word of the first paragraph in Figure 3.3. The word *a* is considerably lighter or faded relative to the characters of the surrounding words. This can be seen at other places in the document too.

Further, the positioning of the characters is not strictly along a single straight horizontal line and the spacing is sometimes uneven. Also, sometimes characters are slightly rotated. For example, the first word of the fourth line in Figure 3.3 is *and*, and the first character of this word (i.e., *a*) is slightly shifted down. One possible explanation for these phenomena could be the mechanical idiosyncrasies of the typewriter.

Furthermore, the fonts of different typewriters vary slightly. Some models of typewriters have the ability to change the font. For example, some models of the IBM Selectric let one change a ball to get a different font style [5].

As mentioned above, since many scanned ETDs were written using typewriters, and since there are many types and variations in the typewriter fonts, we change the font type to resemble the typewriter font. To achieve this, we add the following lines to the beginning of each LaTeX document:

```
\renewcommand\ttdefault{cmvtt}
\renewcommand{\familydefault}{\ttdefault}
```

These lines change the font of the entire document to the CMVTT font in LaTeX. CMVTT is a font available in the LaTeX software libraries that belongs to the Computer Modern font family.

The above commands, however, do not mimic the various quirks of real-life typewriters. Further, the CMVTT font family does not support the bold font [45]. We aim to address these issues as part of future work.

Increased line spacing

We also observed that for scanned ETDs written using typewriters, the line-spacing (the space between two consecutive text lines) in the document is considerably larger than that of a born-digital document. Therefore, we increase the line spacing to 1.5 by adding the following command to the beginning of each LaTeX document:

```
\linespread{1.5}
```

3.3.5 Image-based transformations

To implement the image-based transformations, we use the popular ImgAug [34] software library. We propose the following image-based transformations. For each transformation described below, we mention the software command from ImgAug [34] and its arguments used for implementing the transformation.

Random affine rotation

While scanning the hard-copy of a document, it is possible that the page might be slightly rotated. As a result, in the scanned PDF file, not all pages will be perfectly aligned. Further, this process is not constant; it can happen randomly.

Therefore, for each page of each PDF file in the dataset, we first randomly sample a floating-point number between -5 and +5 from a random number generator following a standard normal distribution. We rotate the image of this page by the number of degrees defined by this number. In other words, we randomly rotate each page, between -5 and +5 degrees.

```
Affine(rotate=(-5, 5))
```

Additive Gaussian noise

A flatbed scanner works by reflecting the light from paper and creating an image of the paper based on the light reflected back. Light reflecting from the surface of the paper is a natural phenomenon. Hence, we add Additive Gaussian Noise to the image to mimic this effect. The parameters of this noise are heuristically chosen using trial-and-error.

```
AdditiveGaussianNoise(scale=(10, 60))
```

Salt-and-pepper noise

Since salt-and-pepper noise is often seen on images caused by sharp and sudden disturbances in the image signal, we use it as our next transform [51].

```
SaltAndPepper(p=0.1)
```

Here, p represents the probability of replacing a pixel by noise. We chose this parameter heuristically.

Gaussian blur

Digital images are limited to a certain resolution as compared to natural objects. This happens because to represent any image in its digital form, there are only a certain number of bytes available to encode it. Therefore, during the process of digitization of any hard-copy, a certain sharpness/resolution is lost.

To incorporate this in our augmentation transformations, we apply Gaussian Blurring. This is basically just image smoothing using a Gaussian Kernel.

```
GaussianBlur(sigma=0.5)
```

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2-y^2}{2\sigma^2}}$$

Sigma is used to generate the Gaussian smoothing kernel using the Gaussian formula.

Linear Contrast

Although today's flatbed scanners are built using advanced technology, they still are not capable of capturing all of the natural colors of a natural object. Therefore, to incorporate this characteristic of flat-bed scanners, we add Linear Contrast.

```
LinearContrast(alpha=1)
```

alpha determines the amount of linear contrast to introduce in the image.

Perspective transform

During the scanning of hard-copies, it is possible that the paper might be stretched. To incorporate this, we implement Perspective Transform.

```
PerspectiveTransform(scale=0.025)
```

Figure 3.4 shows the effect of applying all image-based transformations and 3.5 shows the effect of applying all LaTeX-based transformations on a sample born-digital page of a research paper [22].

Figure 3: An image from the dataset of ?, along with an "interest map" - local saliency computed according to the Itti-Koch model (??). Fixations made by the subjects are overlaid in red. How well does the interest map characterise this fixation pattern? This question is not easily answered by eye, but may be given a more precise meaning in the context of spatial processes.

However, surely other things besides contrast matters - what about average luminance, for example? Couldn't brighter regions attract gaze?

This would lead us to expand our model to include luminance as another spatial covariate, so that the log-intensity function becomes:

$$\log \lambda_i(x, y) = \alpha_0 + \alpha_1 c_i(x, y) + \alpha_2 l_i(x, y)$$

in which $l_i(x, y)$ stands for local luminance. But perhaps edges matter, so why not include another covariate corresponding to the output of a local edge detector $e_i(x, y)$? This results in:

$$\log \lambda_i(x, y) = \alpha_0 + \alpha_1 c_i(x, y) + \alpha_2 l_i(x, y) + \alpha_3 e_i(x, y)$$

It is possible to go further down this path, and add as many covariates as one sees fit (although with too many covariates, problems of variable selection do arise, see ?), but to make our lives simpler we can also rely on some prior work in the area and use *pre-existing, off-the-shelf image-based saliency models* (?). Such models combine many local cues into one interest map, which saves us from having to choose a set of covariates and then estimating their relative weight (although see ? for work in a related direction). Here we focus on the perhaps most well-known among these models, described in ? and ?, although many other interesting options are available (e.g. ? ?, or ?).

The model computes several feature maps (orientation, contrast, etc.) according to physiologically-plausible mechanisms, and combines them into one master map which aims to predict what the interesting features in image i are. For a given image i we can obtain the interest map $m_i(x, y)$ and use that as the unique covariate in a point process:

$$\log \lambda_i(x, y) = \alpha_0 + \alpha_1 m_i(x, y) \quad (3)$$

This last equation will be the starting point of our modelling. We have changed the notation somewhat to reflect some of the adjustments we need to make in order to learn anything from applying model to data. To summarise:

- $\log \lambda_i(x, y)$ denotes the log-intensity function for image i, which depends on the spatial covariate $m_i(x, y)$ that corresponds to the interest map given by the low-level saliency of ?
- α_1 is an image-specific coefficient that measures to what extent spatial intensity can be predicted from the interest map. $\alpha_1 = 0$ means no relation, $\alpha_1 > 0$ means that higher low-level saliency is associated, on average with more fixations, $\alpha_1 < 0$ indicates the opposite - people looked more often at low points of the interest map. We make α_1

Figure 3.4: A sample page from [22] with all image-based transformations applied.

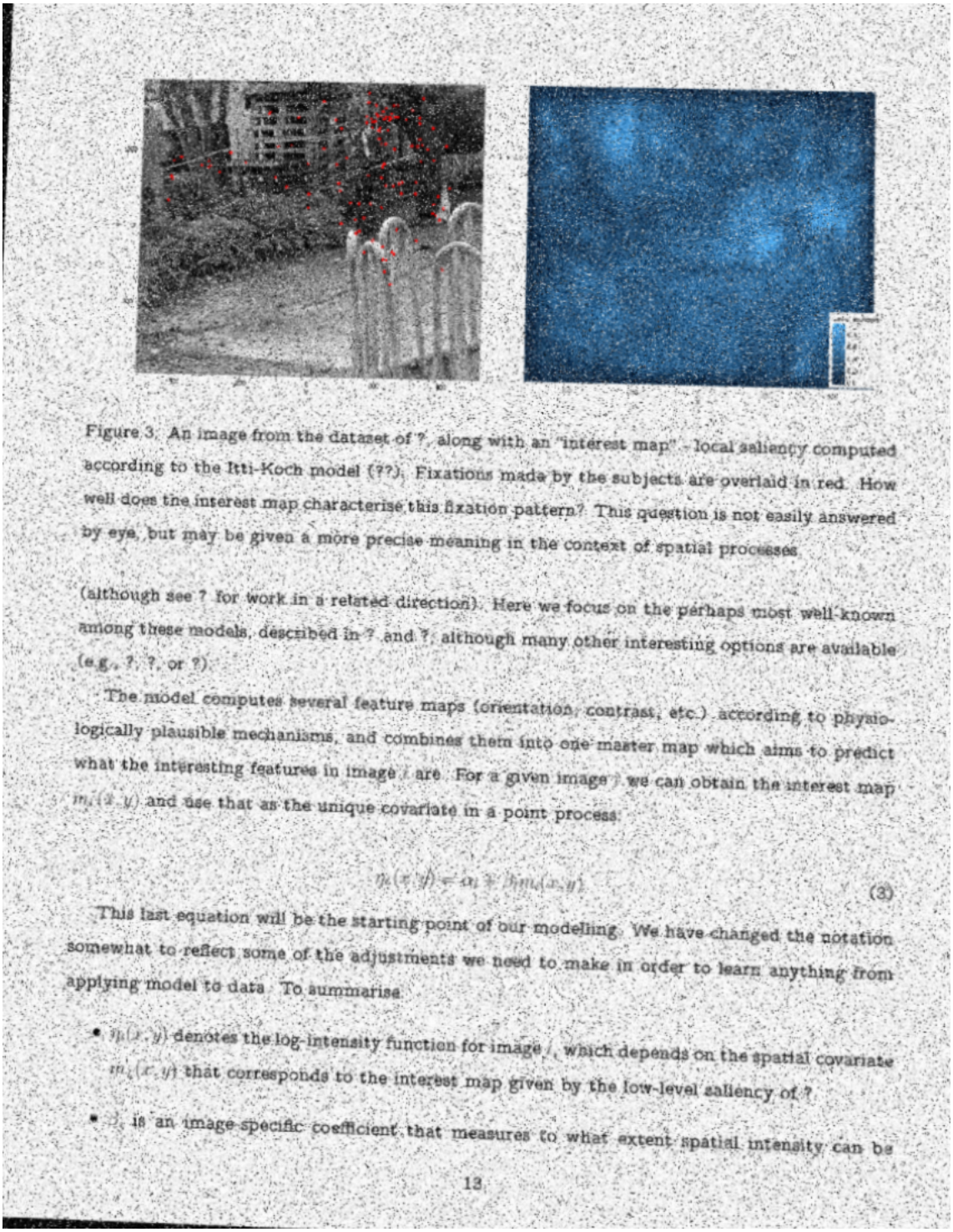


Figure 3. An image from the dataset of ?, along with an "interest map" - local saliency computed according to the Itti-Koch model (??). Fixations made by the subjects are overlaid in red. How well does the interest map characterise this fixation pattern? This question is not easily answered by eye, but may be given a more precise meaning in the context of spatial processes

(although see ? for work in a related direction). Here we focus on the perhaps most well-known among these models, described in ? and ?, although many other interesting options are available (e.g. ?, ?, or ?).

The model computes several feature maps (orientation, contrast, etc.) according to physiologically plausible mechanisms, and combines them into one master map which aims to predict what the interesting features in image i are. For a given image i we can obtain the interest map $m_i(x, y)$ and use that as the unique covariate in a point process:

$$\eta_i(x, y) = \alpha_i + \beta_i m_i(x, y) \quad (3)$$

This last equation will be the starting point of our modelling. We have changed the notation somewhat to reflect some of the adjustments we need to make in order to learn anything from applying model to data. To summarise

- $\eta_i(x, y)$ denotes the log-intensity function for image i, which depends on the spatial covariate $m_i(x, y)$ that corresponds to the interest map given by the low-level saliency of ?
- β_i is an image specific coefficient that measures to what extent spatial intensity can be

13

Figure 3.5: A sample page from [22] with both image-based and LaTeX-based transformations applied.

3.4 Generating labels

All of the above transformations for augmenting the data are done in conjunction with the original Deepfigures pipeline. Figure 3.6 provides a better overall view of the modified pipeline.

In Figure 3.6, the step named "Add markup for LaTeX augmentations" is our modification to the Deepfigures pipeline for adding the LaTeX-based transformations.

Similarly, the step named "Apply image-based augmentations using ImgAug" is another modification to the Deepfigures pipeline for adding the image-based transformations.

We use PDFLaTeX for compiling all LaTeX documents. Specifically, we use the TexLive [11] distribution.

3.5 System design to run our pipeline at scale.

Training a deep learning model often involves dealing with a large amount of data. The typical process of training a model involves storing, processing, and reading this data. As the size of this data grows, it becomes increasingly difficult to deal with it. Problems such as storage, pre-processing, and reading it are often encountered at scale.

In this work, we train our deep learning model on the arXiv dataset, which contains over 1 TB of data. Therefore, in this section, we describe how we designed a scalable pipeline to pre-process this data for training the model in real-time.

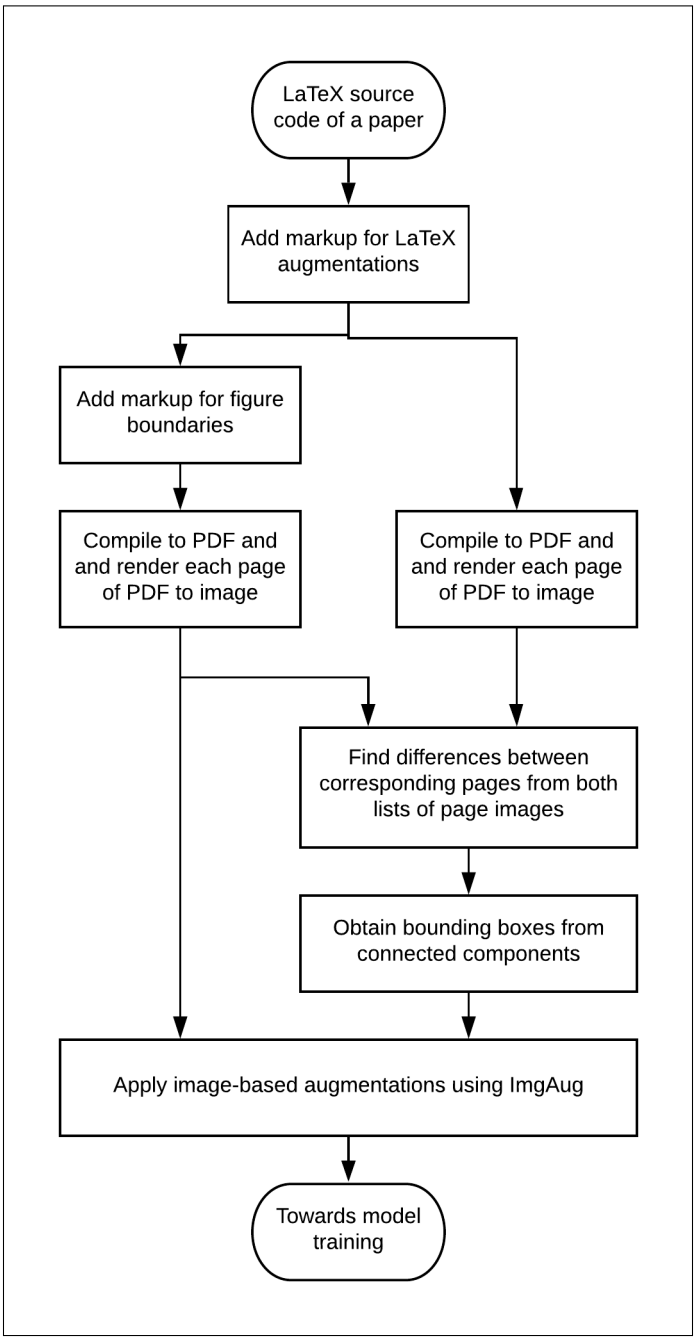


Figure 3.6: The modified version of the Deepfigures pipeline for our experiment.

3.5.1 Background

Dataset description

As described in Chapter 2, the goal is to train a deep learning model (ResNet-101 + Overfeat) for predicting the bounding boxes around the figures of scholarly documents. The training data for this is the arXiv dataset available at [58] along with its description. The entire dataset is about 1.2 TB on disk. It consists of about 2600 tar files. Each tar file is not more than 500 MB in size. Within each of these files, there are multiple tar.gz files, each of which contains the LaTeX source code for one scholarly paper.

Training pipeline

The input to the deep learning model is an image of each page of each scholarly paper along with the coordinates of the bounding boxes for all figures on that page, if any. This is obtained using the following steps. A detailed description of these steps can be found in [54].

- For each tar file, unzip it.
- For each tar.gz within the tar file, unzip it.
- Compile the LaTeX source code and convert each page of the resultant PDF file into images.
- Modify the LaTeX source code such that it draws bounding boxes around figures after compiling. Repeat the process in the previous step to obtain the images.
- Perform a pixel-by-pixel subtraction of the corresponding images generated in the above two steps to obtain the coordinates of bounding boxes.

3.5.2 Challenges

Executing the above-mentioned steps at scale introduces the following engineering challenges:

Disk space

An obvious way to achieve this is by first unzipping and processing the entire 1.2 TB dataset, and then starting the training of the deep learning model. However, unzipping the entire 1.2 TB arXiv dataset and performing all of the steps in the pipeline results in a total disk space of about 30 TB. Obtaining such a large amount of disk space is costly and might not always be feasible.

Processing time

The data pre-processing steps in the pipeline involve unzipping of huge files, compiling LaTeX source code and doing image processing operations. These steps make the pipeline computationally expensive. Using a single CPU core for pre-processing the entire 1.2 TB of dataset will take a long time and thus will be prohibitively slow for training (because the GPU will have to wait for the single CPU to generate the data).

The approach mentioned in the Deepfigures paper [54] processes the data in parallel using multiple cores. However, it needs to store the entire 30 TB of processed data on disk before the training can be started. As a result, it needs a lot of disk space and that becomes a bottleneck.

3.5.3 Proposed system

We propose a solution which will allow us to improve the performance of the pre-processing pipeline when used in conjunction with the training of the deep learning model.

Main intuition

We know that the pre-processing pipeline and the training of the model have different kinds of computing requirements. In other words, the pre-processing pipeline uses only the CPU (multiple cores, if parallelized) and the training of the deep learning model mostly uses the GPU. We leverage this fact to do both the pre-processing and the training simultaneously. We design our code in such a way that multiple CPU cores will pre-process the data in parallel and will supply it to the deep learning model. This ensures that neither the CPU nor the GPU has unused capacity during the entire process.

Implementation

Figure 3.7 shows the system diagram of our implementation. We declare N processes for loading and pre-processing the data from the disk. Each of these processes will work on one single tar file which it will pick up from a process-safe queue. At the start of the program, for the arXiv dataset, this queue should contain about 2600 items. Each item will be a file pointer (e.g., file path) to one of the 2600 tar files in the arXiv dataset.

Each process will read the .tar file it is currently processing, unzip it, and for each paper tar.gz will follow the process mentioned in Figure 3.6. Once it is done processing, it will pass the processed output to the main training process for training. Each of these worker processes will be allocated a separate temporary disk space for operations such as unzipping files, etc. The life-cycle of these temporary directories is entirely managed by each of these

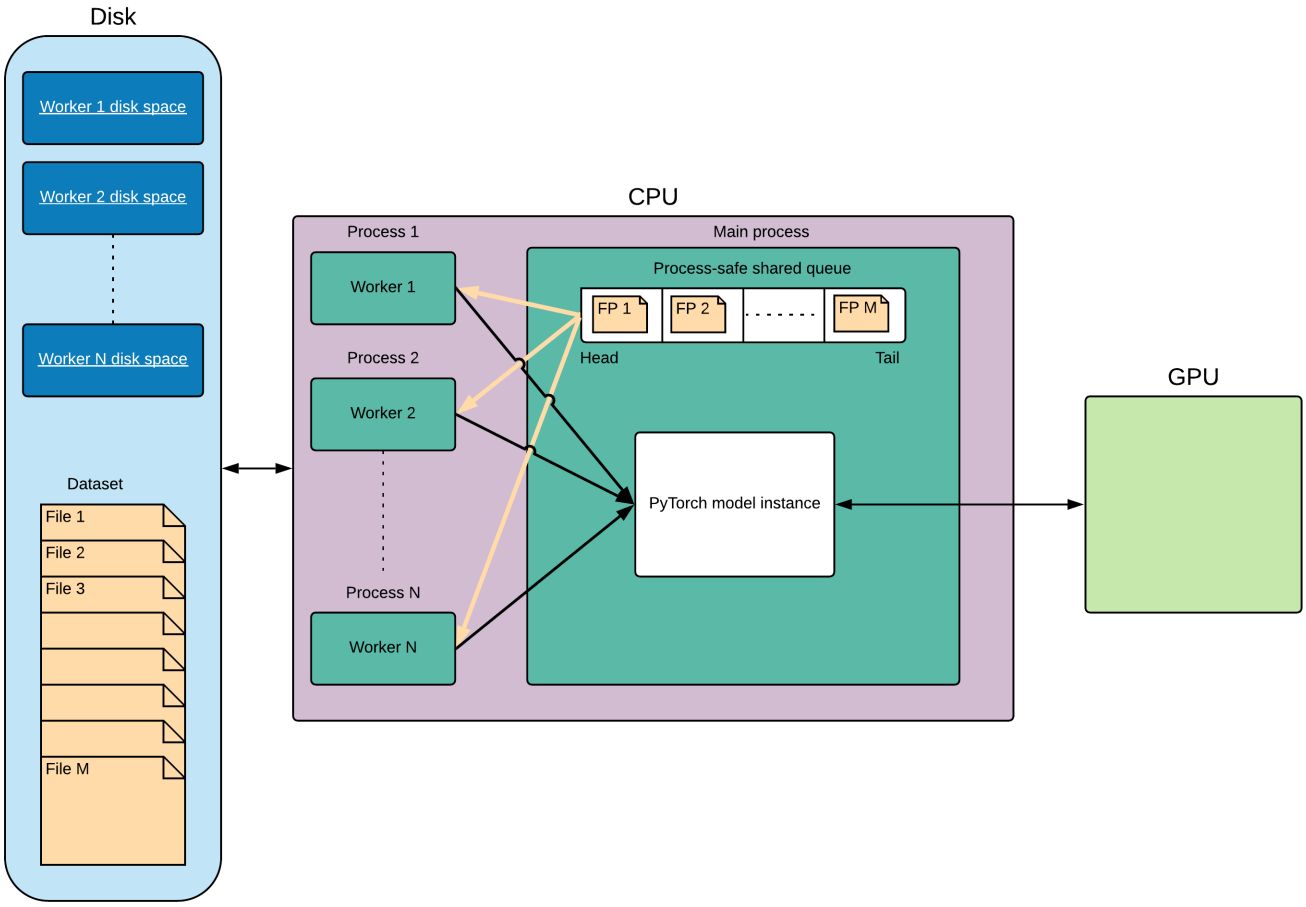


Figure 3.7: System diagram of our system for running the modified Deepfigures pipeline at scale.

individual worker processes. They are cleaned up when the process ends.

Possible shortcomings

One of the limitations of this approach is that the processed dataset is not being stored. It needs to be re-generated each and every time we need to train. However, this limitation can be easily overcome by implementing a disk-backed caching layer. This way, every time any part of the dataset is processed, its processed version gets stored in the cache for future

use. Only the necessary data can be stored in this disk-backed cache in a compressed form, thereby alleviating storage space limitations for the cache.

3.6 Gold standard

We use the data generated using our data augmentation pipeline for training our model. A common practice is to split the entire data available into train, validation, and test sets to benchmark the performance of the model. Out of these, the validation set is used to fine-tune or choose the best model during training while the test set is used to get a final unbiased estimate of the model. However, the shortcoming of this method is that if there is any noise in the available data, the validation and the test sets would have noise too. This would lead to inaccurate performance numbers. To solve this problem, we create a gold standard dataset. The gold standard dataset will act as the ground-truth and thus can be used to get a reliable estimate of model performance.

3.6.1 MIT DSpace

MIT's DSpace [15] is used in its institutional repository. Among other scholarly documents, that contains collections of ETDs authored by MIT students.

Each ETD in these collections first exists as a hard-copy and is then scanned into DSpace. Since we want to create a gold standard dataset whose feature distribution should closely match the distribution of scanned ETDs, these scanned ETDs are useful to us.

Another important reason for using MIT's DSpace for our gold standard is that these ETDs are organized into separate department-wise sub-collections. Further, the ETDs from each department are categorized into the type of degree (e.g., Masters, Ph.D., Bachelors, etc.).

This helps reduce the bias in the type of figures in the gold standard, if any. Further, this also enables us to do an analysis on different groupings such as by department or degree.

Lastly, each ETD also has accompanying metadata. This contains information such as the date of publication, author name, etc. This helps us filter the ETDs based on the year in which they were published.

3.6.2 Collecting ETDs

URLs used for crawling

Each ETD in MIT's DSpace website is uniquely represented by its handle. A sample handle is *1721.1/42427*. Given the handle of any ETD, its PDF file and its metadata can be easily accessed through the URL:

```
https://dspace.mit.edu/handle/1721.1/42427?show=full
```

In fact, any resource in the website (e.g., ETD, collection, sub-collection, etc.) is represented by a unique handle. We leverage this fact to crawl the website for a given department and obtain the handles of the associated ETDs. Given the handle of a department, the list of its ETD handles can be obtained in a paginated fashion through its browse page. For example, for the Department of Electrical Engineering and Computer Science, the handle is *1721.1/7599* and its browse URL is:

```
https://dspace.mit.edu/handle/1721.1/7599/browse
```

Crawler

We write a simple crawler script in Python using web scraping tools like LXML [6] to download the handles of all ETDs using the browse URL of each department. Further, for each ETD handle, we use its URL to download its PDF file and its metadata. Although the PDF file is relatively easy to download, the metadata is in HTML format. Therefore, to convert it to a format better suited for querying, we used the *read_html()* function in Pandas [46] to convert it to JSON.

Rate limits

MIT's DSpace website implements a rate limit for the number of requests an IP can make to it. We figured this out when we ran our crawler scripts to download the ETDs and the metadata. We were periodically getting HTTP 429 as the response code when we exceeded a certain number of requests from a given IP address. To solve this issue, we added retry mechanisms in our crawling script. As a result, whenever HTTP 429 was the response, we used to sleep our crawler for 5 seconds and then try again. This way, we prevented MIT's DSpace website from getting overloaded, which perhaps was the intention behind the rate limiter.

3.6.3 Sampling ETDs

Out of the downloaded ETDs, we randomly sampled ETDs with some constraints. The date of issue should be pre-1990, and at-most one ETD should be sampled from each sub-community (doctoral theses, masters theses, bachelor's theses, etc.) within each department. After accounting for empty sub-communities, our sample contained a total of 70 ETDs 3.8.

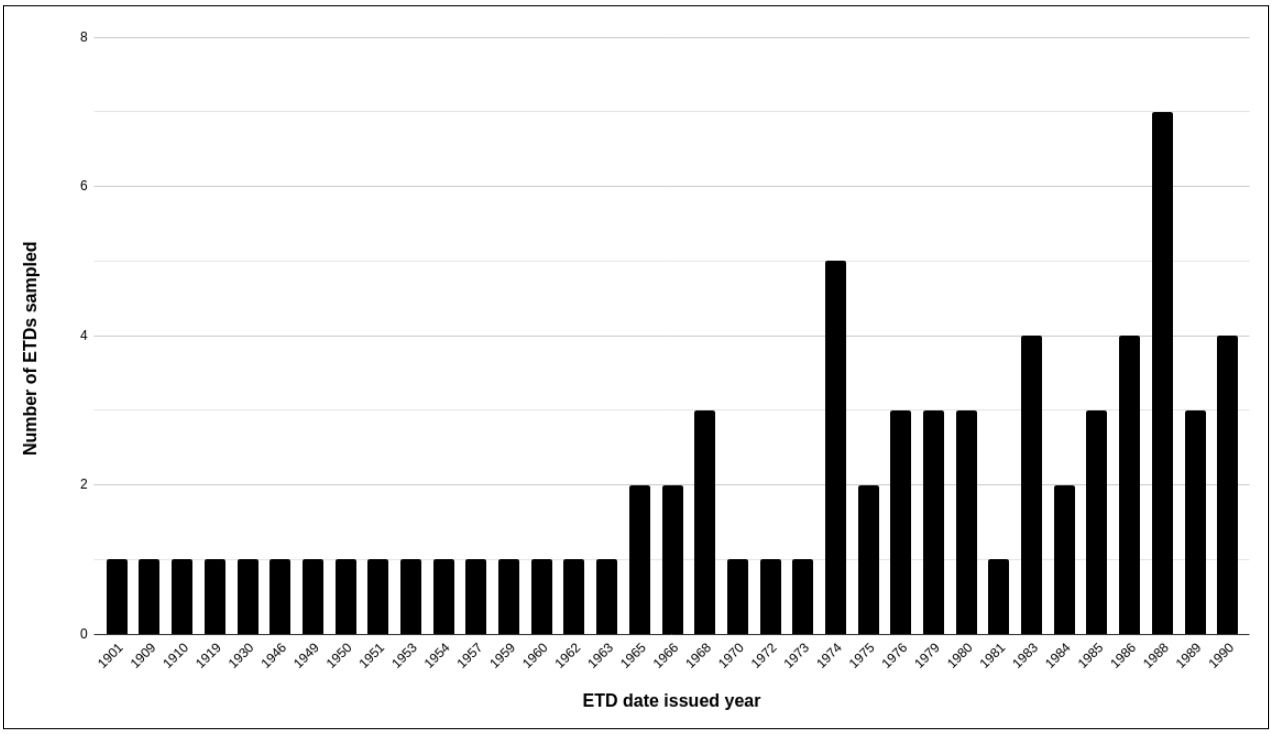


Figure 3.8: Year-wise distribution of ETDs sampled for the gold standard dataset.

3.6.4 Labelling ETDs

After collecting the ETDs for this dataset, we converted each ETD into the images of their pages at 100 DPI. We did not change the original aspect ratio of the pages. In this process, a total of 10182 images of pages were obtained. We used the VGG Image Annotator (VIA) [27, 28] tool to manually label these images with bounding boxes around figures. The following guidelines were used when carrying out this labelling:

- Some papers had a lot of software code directly written in the PDF file. This code was not labelled.
- Table of Contents was labelled.
- Captions for both figures and tables were labelled.

- List of figures was labelled.
- List of tables was labelled.
- Bibliography was not labelled.
- Mathematical equations (including matrices) were not labelled.
- For screen-captures (including newspaper cut-pastes, which had figures in them), the individual figures within the figures were labelled. The encompassing figure was not labelled. No nested labelling was done.

A total of 3375 figures were labelled across the entire dataset. The total number of figures across each of the 70 ETDs is shown in Figure 3.9.

A screenshot of the VGG Image Annotator tool while creating labels for the gold standard dataset appears in Figure 3.10.

Note: For manually labelling the gold standard dataset, the bounding boxes were drawn such that the figures and tables as well as their captions were circumscribed by the bounding box. This was done for the following two reasons:

- The bounding boxes generated by the Deepfigures pipeline exhibit a similar behaviour, wherein, the bounding box circumscribes both the figure/table and its caption (see Figure 2.8).
- As per our intuition, if there is a future need to separately extract the captions from figures/tables, it could be easier to do so from an already extracted sub-region of the page as opposed to extracting it from the entire page.

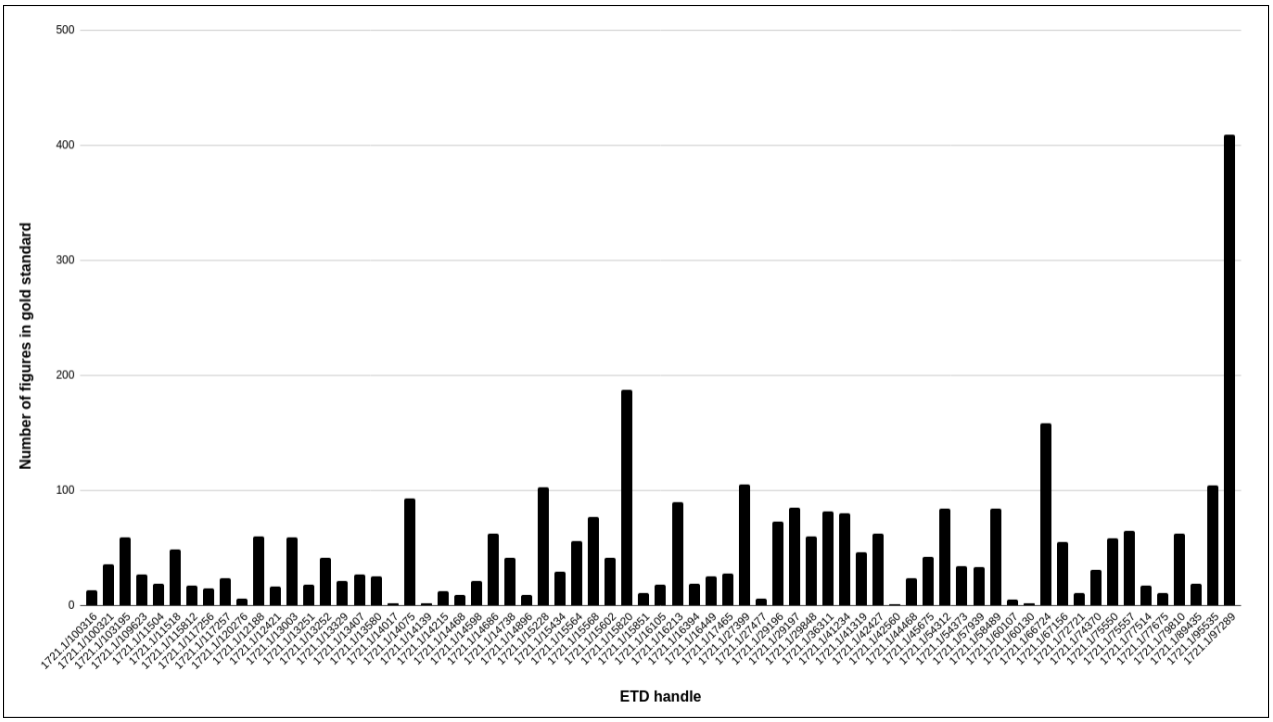


Figure 3.9: Figure count per ETD in the gold standard dataset. X-axis shows the ETD handles from [15].

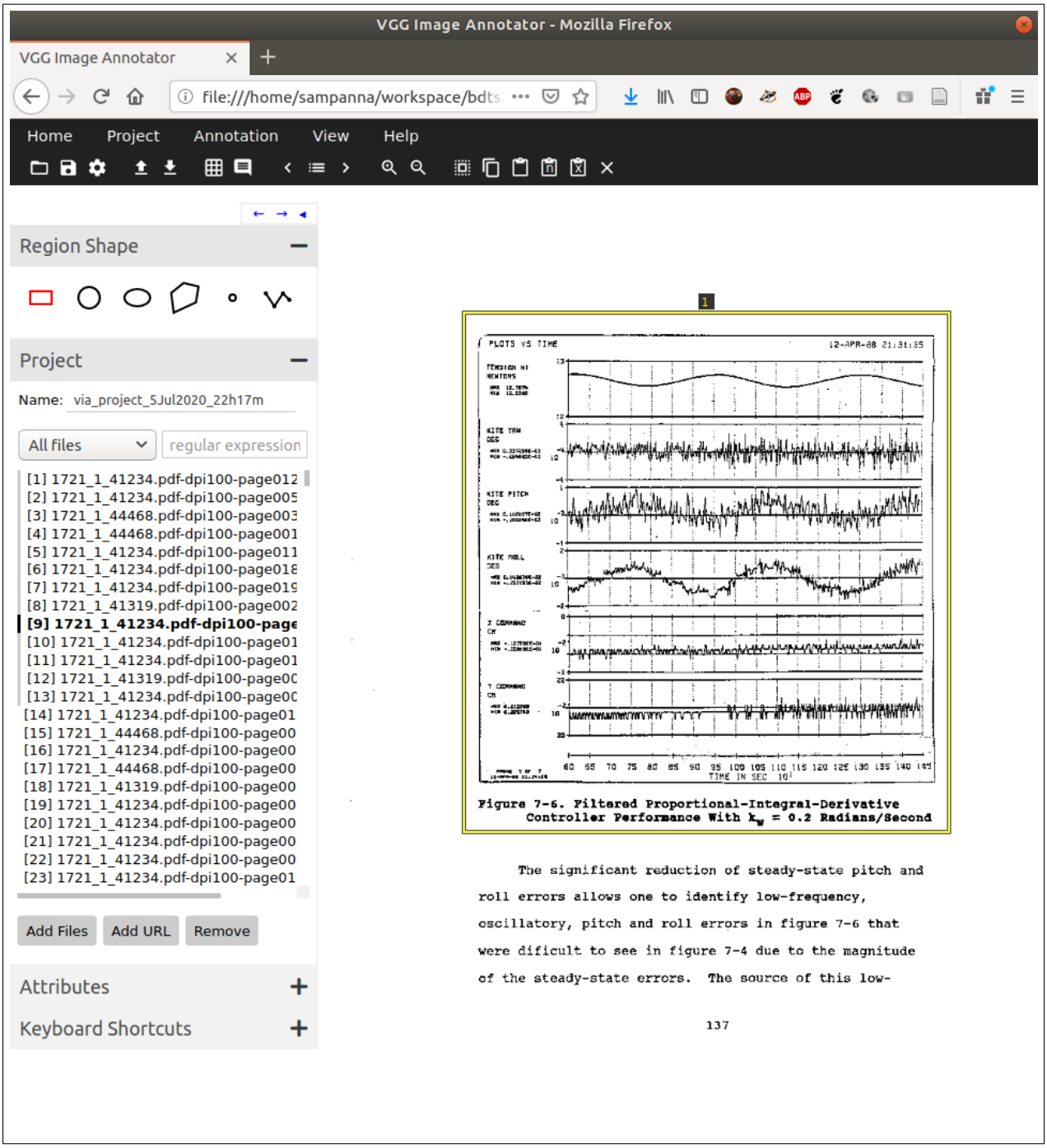


Figure 3.10: A screenshot of VGG Image Annotator tool [27, 28] during creating labels for the gold standard dataset.

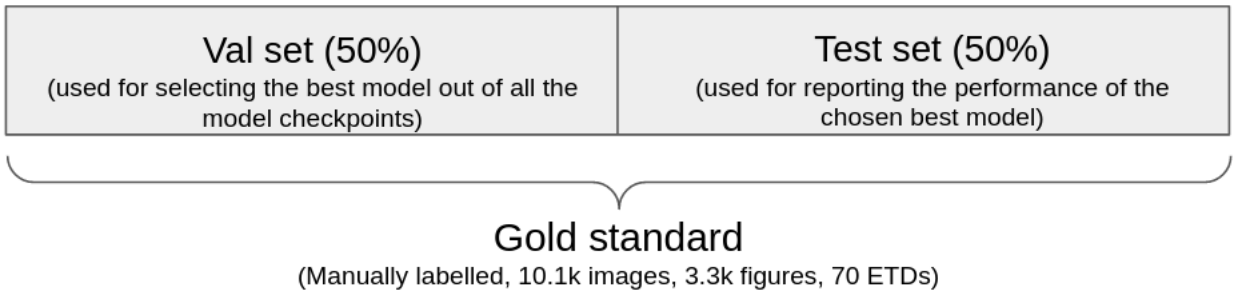


Figure 3.11: Validation and test splits of the gold standard dataset

3.6.5 Validation and test splits

We split the gold standard dataset into two equal halves after shuffling it randomly. The first half will be used as the validation set for fine-tuning or choosing the best model during training. The second half will be used as the test set for reporting the final unbiased estimate of the model. In the rest of this thesis, we use these two sets for reporting our results (Figure 3.11).

Chapter 4

Results and discussion

In this chapter, we describe the various experiments we conducted as a part of this work, in chronological order.

4.1 Experiment 1

4.1.1 Experimental setup

The Deepfigures [54] model was trained on the data generated from two datasets — arXiv and PubMed. However, the size on disk for these two datasets is quite large. Therefore, training a deep learning model, albeit using a GPU, takes a long time (sometimes days). Therefore, in this experiment our aim is to do a proof of concept (POC) on a smaller dataset.

In this experiment, we train the model on the data and labels generated using our modified pipeline. However, instead of training on the entire dataset, we use a small subset of the dataset.

Sampling a subset of the arXiv dataset

In order to choose the subset of the dataset, we limit ourselves to the arXiv dataset and ignore the PubMed dataset. This was done because the data augmentations described in

Section 3.3.5 and Section 3.3.5 can be applied only if the LaTeX source of the paper is available.

We further reduce the size of the training data by randomly sampling from the arXiv dataset. One of the ways to sample from the arXiv dataset is to randomly drop a few training images and their labels when running the pipeline described in Figure 3.7. Although this method provides a higher degree of randomness, the limitation of doing so is that we will still have to process the entire dataset. This will lead to unnecessary computation.

Therefore, we optimize the procedure by randomly sampling 50 out of the 2600 tar files from the arXiv dataset. The limitation of using this method is that we don't randomly sample data within a tar file. In other words, if we choose a tar file, all of the papers in it are counted in our sample set. However, the benefits far outweigh the limitations while maintaining a good degree of randomness within the sample. Using this method, we don't have to process the entire dataset — just what is sampled. Further, the training process does not need access to the entire dataset, which can alleviate some disk space problems.

Training

We train the model using the modified Deepfigures pipeline we proposed (Figure 3.6). Specifically, in this experiment, we use all augmentations mentioned in Section 3.3.5 and Section 3.3.5.

We use the same hyper-parameters as the ones used in [54]. We train it for 100,000 steps with a batch size of 1.

4.1.2 Results

Table 4.1 shows the results of the experiment. We trained two models. The first was trained using only the image-based transformations for data augmentation. The second was trained with both image and LaTeX-based transformation. We tested them by comparing the predictions with a manually labelled scanned sample ETD [25]. This ETD was chosen randomly from Virginia Tech's VTechWorks digital library since the gold standard dataset using MIT ETDs was not yet created during the time of conducting this experiment. The ground truth of this ETD contains 26 figures and no tables. This ETD also contains some non-textual pages with blue background representing some kind of blueprints. However, we exclude such pages for the purposes of this evaluation.

Table 4.1: Performance of the original Deepfigures model as compared to the re-trained models when run on a scanned ETD from VTechWorks [25]. (TPs=True Positives, FPs=False Positives, FNs=False Negatives)

Model	TPs	FPs	FNs	Precision	Recall	F1
Deepfigures	0	29	26	0	0	0
Ours (image-based transformations)	7	16	15	0.30	0.318	0.309
Ours (all transformations)	10	15	16	0.4	0.385	0.392

From Table 4.1, we can see that as we add more augmentations, the true positives increase and the false positives and the false negatives decrease. This clearly shows that our data augmentation techniques improve the model performance on the scanned ETD we tested on. However, to get more robust numbers, we decide to conduct more experiments using more scanned ETDs and better evaluation metrics.

4.2 Experiment 2

4.2.1 Experimental setup

We lacked two things in the previous experiment. Firstly, we didn't have strong evaluation metrics for measuring the performance of our model. In other words, although we computed the precision, recall, and F1-scores, we did not define mathematically how we deem a predicted bounding box as a true positive, etc. We describe this in more detail in Section 4.2.1. Secondly, we evaluated it only on a single scanned ETD. Therefore, we address these two limitations in this experiment.

More importantly, to train more models and compare them with the original Deepfigures model, we need to know how the original Deepfigures model performs on the gold standard. For Deepfigures [54], the authors have released the trained model weights. We use these weights to run inference for the validation and test sets of the gold standard dataset on that model.

Evaluation metrics

The Deepfigures [54] model outputs a set of bounding boxes for each figure it detects in the image. We match these predicted bounding boxes to the true bounding boxes in order to minimize the total Euclidean distance between the centers of paired bounding boxes. As mentioned in [26, 54], this is an instance of the linear assignment problem and can be efficiently solved using the Hungarian algorithm [39]. Once the predicted boxes have been matched with the true boxes, we deem a predicted box as correct if its intersection over union (IOU) with the true box is greater than or equal to 0.8 (i.e., True positive), incorrect if less than 0.8 (False Positive) [54]. If the model fails to detect a box which it should have

detected, we deem it as a False Negative. In other words, if the model predicts a box even though there is no figure present inside it, we deem it as a False Negative [37]. Using these three metrics and the formulae mentioned in Section 2.1.9, we can find the precision, recall, and F1 score.

Evaluating on gold standard

As mentioned in Section 3.6, we create the gold standard to better evaluate model performance. During the training, we store multiple checkpoints of the model at regular intervals. These checkpoints consist of the model weights, the definition of the model architecture, and some metadata. In other words, these checkpoints contain all the necessary information needed to re-instantiate the model and it's weights at the state in which the checkpoint was created. We use the validation set to choose the best performing model from all the checkpoints, and we use the test set to report the performance of this selected best model.

4.2.2 Results

Table 4.2 shows the performance of the original Deepfigures model on the gold standard dataset. When we compare the performance of Deepfigures in Table 4.1 with its performance in Table 4.2, we can see the improvement. According to Table 4.1, the Deepfigures model is not able to detect any True Positives. However, according to Table 4.2, the Deepfigures model does detect some figures. One possible explanation for the better performance of Deepfigures in this experiment is that we evaluate it only on a single ETD, as discussed in Section 4.1, and perhaps that ETD is an outlier. This highlights the importance of using a well-sampled and balanced gold standard dataset for evaluating models.

In Section 1.2, RQ1 was posed in which we aim to explore the performance of existing

methods for figure extraction from scanned ETDs. In this experiment, we evaluated the performance of Deepfigures on our gold standard dataset using metrics such as precision, recall, and F1-score, thereby answering RQ1.

Table 4.2: Performance of the original Deepfigures model on the gold standard dataset. (TPs=True Positives, FPs=False Positives, FNs=False Negatives)

Model	TPs	FPs	FNs	Precision	Recall	F1
Deepfigures	1005	1227	1143	0.450	0.468	0.459

4.3 Experiment 3

4.3.1 Experimental setup

As discussed in Section 4.1, we trained the model on the data augmented using all the transformations (i.e., image-based as well as LaTeX-based). However, since we have chosen these transformations heuristically and not based on quantifiable metrics, it is possible that not all of these transformations are causing the performance of the Deepfigures model to improve.

Finding 'helpful' transformations

Each transformation in the list mentioned in Section 3.3.4 or Section 3.3.5 can either be disabled or enabled. The total number of possible combinations for this is 2^n where n is the number of transformations possible. Training 2^n models for a significantly large number of transformations might not be computationally feasible. Therefore, we propose a leave-one-out ablation study to get a better idea about which transformations are actually helpful.

Leave-one-out ablation study

In an leave-one-out ablation study, we train n different models in parallel, where n is the total number of possible transformations. In each of these models, we keep all of the hyper-parameters constant, except the list of transformations to apply. For the n-th model, we disable the n-th transformation and leave the remaining enabled. Based on the results, if the n-th model performs poorly, we can say that disabling the n-th transformation has worsened the performance and therefore, enabling that transformation contributes positively towards the model performance.

A note about the leave-one-out ablation study

Although the leave-one-out ablation study gives us some idea about the performance of each transformation, it does not give the entire picture. For instance, it is possible that two transformations performing well on their individual ablation studies could perform worse when used together. This can be further extended for combinations of more than 2 transformations. Therefore, we do not claim that the leave-one-out strategy gives us the most optimal combination of transformations. However, it does help in getting a general idea for weeding out the worst performing transformations, and therefore helps in improving the overall performance of the model.

4.3.2 Results

We trained 11 deep learning models as described in Section 4.3.1.

Interpreting the plot of F1-score vs. training step

During the training of these models, we saved periodic checkpoints of the model weights. We ran inference on the validation set for each of these checkpoints and plotted the F1-score against the training step. One such plot of the F1-score vs. the training step can be seen in Figure 4.1. The plot in Figure 4.1 is for one of the ablation studies in which Additive Gaussian Noise was excluded. The learning rate used for training this model is plotted against the training step in Figure 4.2.

From Figure 4.1, we observe that the very first value of the F1-score is about the same as that of Deepfigures. This is expected because without any training, the model is identical to Deepfigures.

As the training progresses for the first few hundred steps, the F1-score varies quite drastically. This change can be attributed to the relatively high learning rate in the initial part of the training.

In the latter part of the training, as the learning rate decreases, the F1-score stays relatively stable. The F1-score frequently surpasses the F1-score of Deepfigures before slightly going down again. It is during this phase of the training process that the peak F1-score is achieved.

From these observations, it can be said that for this experiment, we should start from a lower learning rate since it is at this stage that we see the F1-score improve. Therefore, we note the task of training with lower learning rate as part of future work.

Interpreting the ablation study results

In Table 4.3, for each trained model, we choose the model's checkpoint which was performing best on the validation set of the gold standard. Then, we reported its accuracy on the test

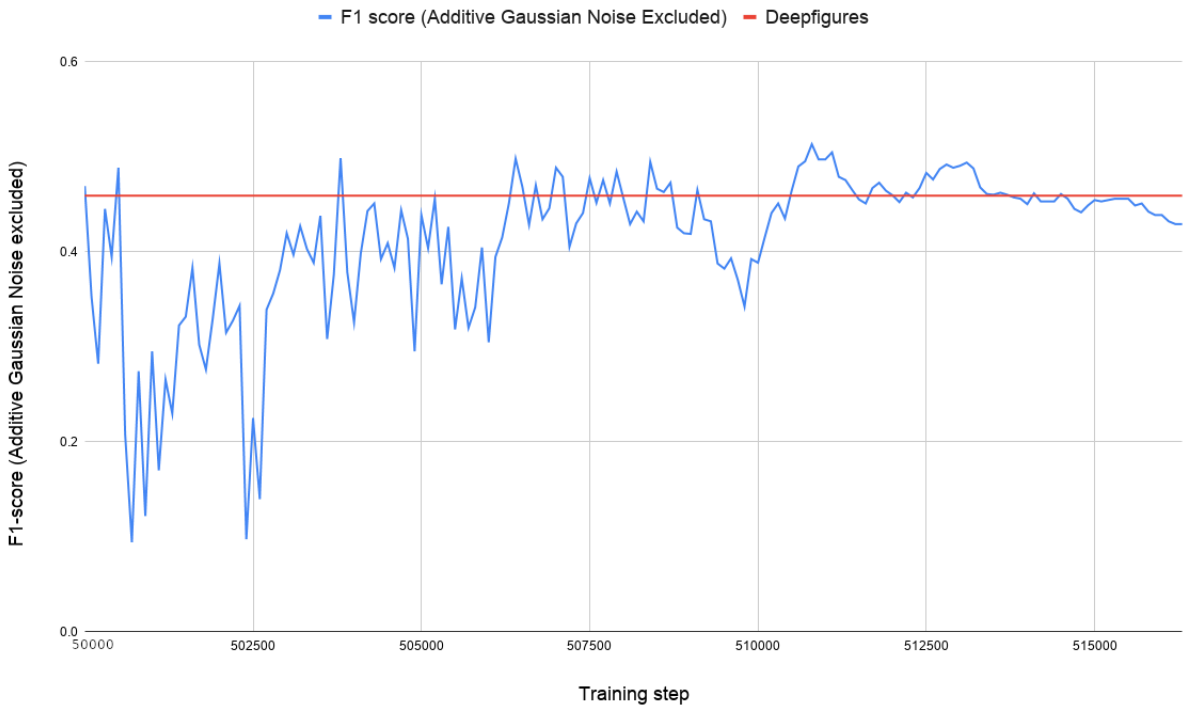


Figure 4.1: Plot of F1-score evaluated on the validation set vs. the training step when trained as part of the 24-hour ablation study in which Additive Gaussian Noise was excluded. The red line indicates the performance of Deepfigures.

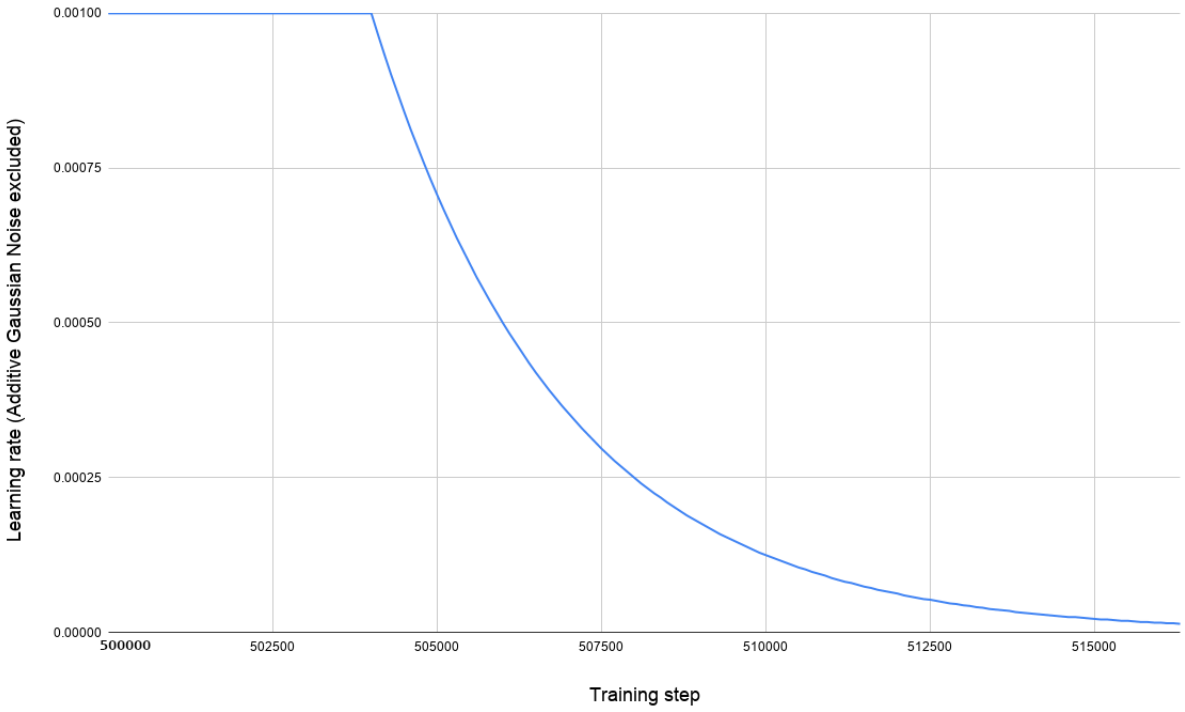


Figure 4.2: Plot of the learning rate vs. the training step when trained as part of the 24-hour ablation study in which Additive Gaussian Noise was excluded.

Table 4.3: Performance of the original Deepfigures model on the gold standard dataset compared to the models trained as part of our leave-one-out ablation study. All models were trained for 24 hours each. (TPs=True Positives, FPs=False Positives, FNs=False Negatives)

Model	TPs	FPs	FNs	Precision	Recall	F1
Deepfigures	1005	1227	1143	0.450	0.468	0.459
Ours (All enabled)	619	506	569	0.550	0.521	0.535
Ours (Additive Gaussian Noise)	561	465	668	0.547	0.456	0.498
Ours (Affine)	577	530	587	0.521	0.496	0.508
Ours (Gaussian Blur)	506	619	569	0.450	0.471	0.460
Ours (Linear Contrast)	630	498	566	0.559	0.527	0.542
Ours (Perspective Transform)	597	539	558	0.526	0.517	0.521
Ours (Salt and Pepper)	686	509	499	0.574	0.579	0.576
Ours (Line spacing 1.5)	614	737	343	0.454	0.642	0.532
Ours (Typewriter font)	566	476	652	0.543	0.465	0.501

set of the gold standard. If a model selected using this method performs well on the test set, we can safely say that it will perform well on any new data it encounters.

We draw the following observations from this table:

- We observe that the F1-scores of almost all of our models surpass the F1-score of the original Deepfigures model. This observation supports answering RQ2 affirmatively. It, however, is not enough to completely support an affirmative response to RQ2 since a single set of observations from one experiment may not be statistically significant.

- The model in which Gaussian Blur was disabled has a score of 0.460. This is close to the F1-score of the original Deepfigures model (0.459). In all of the other models, Gaussian Blur was enabled, and their performance is significantly higher than the original Deepfigures model. Therefore, it is likely that Gaussian Blur is the most 'helpful' transform because disabling it decreases the performance, while enabling it improves performance. Again, since this is just a single experiment, we cannot say

that this is conclusive.

Since the observations made above are not enough to answer RQ2 confidently, we investigate further in the next experiment.

4.4 Experiment 4

4.4.1 Experimental setup

This experiment is similar to the one discussed in Section 4.3, where each model was trained for 24 hours. However, it is possible that 24 hours might not be sufficient for all of the models to train fully. Further, getting an additional set of observations could help us make a more robust argument. Therefore, in this experiment, we re-run the ablation studies with the same hyper-parameters but for 72 hours each.

4.4.2 Results

Table 4.4 shows the results when the ablations studies were conducted for 72 hours.

We make the following observations from Table 4.4:

- We observe that almost all of our models significantly out-perform the original Deepfigures model. This observation is in agreement with the one made in Section 4.3.2. Remember that RQ2 (posed in Section 1.2) aims to explore whether using data augmentation helps to improve the model performance. Therefore, using the observation in Section 4.3.2 and the one made here, we can say with greater confidence that using

Table 4.4: Performance of the original Deepfigures model on the gold standard dataset compared with the models trained as part of our leave-one-out ablation study. All models trained for 72 hours each. (TPs=True Positives, FPs=False Positives, FNs=False Negatives)

Model	TPs	FPs	FNs	Precision	Recall	F1
Deepfigures	1005	1227	1143	0.450	0.468	0.459
Ours (All enabled)	604	482	608	0.556	0.498	0.526
Ours (Additive Gaussian Noise)	613	448	633	0.578	0.492	0.531
Ours (Affine)	589	407	698	0.591	0.457	0.516
Ours (Gaussian Blur)	642	510	542	0.557	0.542	0.550
Ours (Linear Contrast)	602	460	632	0.567	0.488	0.524
Ours (Perspective Transform)	560	739	395	0.431	0.586	0.497
Ours (Salt and Pepper)	625	503	566	0.554	0.525	0.539
Ours (Line spacing 1.5)	705	594	395	0.542	0.641	0.588
Ours (Typewriter font)	641	386	667	0.624	0.490	0.549

data augmentation helps improve the performance of the original Deepfigures model on scanned ETDs, thereby answering RQ2.

- We further observe that the F1-score of the model in which Gaussian Blur is disabled is no longer the lowest performing model. In fact, it significantly outperforms the original Deepfigures model. This is in contrast to the observation made in Section 4.3.2.

- Further, in Table 4.3, the F1-score of the model in which Additive Gaussian Noise is disabled (0.498) is lower than the F1-score of the model in which Affine transform is disabled (0.508). However, in Table 4.4, the F1-score of the model in which Additive Gaussian Noise is disabled (0.531) is higher than the F1-score of the model in which Affine transform is disabled (0.516). Because of these contrasting observations, we cannot yet conclude that ablation studies aid in identifying the 'helpfulness' of the various data transforms, and hence more investigation is recommended.

4.5 Experiment 5

4.5.1 Experimental setup

We decided to train the model on the gold standard data. The dataset consists of about 10k images with 3.3k labels for figures. Instead of doing a random 50-50 split for training and testing sets, we decide to do a random 80-20 split in order to train the model on as much data as possible.

In the previous experiments, we trained on the data generated using the modified Deepfigures pipeline and we were evaluating on the gold standard. However, for this experiment, since we are training on the gold standard data, we decide to not have separate validation and test sets.

Further, unlike the previous experiments, instead of running the inference on the entire validation and testing set after each checkpoint, we run inference only on a single batch from the testing set.

4.5.2 Results

Figure 4.3 is a plot of the testing accuracy on the y-axis and the training-step on the x-axis. The training step starts from 500,000 because the original Deepfigures model [54] was trained on 500,000 steps, and we use it for weight initialization.

In other words, when the authors of [54] started training the Deepfigures model, its weights were randomly initialized. They trained it for 500,000 training steps (training batches). Since we use the model architecture and the training weights released [54] for initializing our weights, in a way we are continuing their training from 500,000-th step onward (albeit on

augmented data).

Further, the plot of test accuracy is smoothed using an exponential moving average filter for better interpretation. In a moving average filter, each data point in the data window is regarded equally important when calculating the average. In an exponential moving average filter, more emphasis is placed on the recent data by computing the weights using an exponential function [7].

In Section 1.2, we pose RQ3 which aims to explore whether the performance of the original Deepfigures model can be improved by training it on the gold standard dataset. As we can observe from the plot, the test accuracy continues to drop, and does not improve. This suggests that the performance of the model for extracting figures is slowly worsening when trained on the gold standard dataset, thereby answering RQ3.

We speculate that this could have happened because of, but not limited to, the following two reasons:

- Because of the weight initialization, the starting point of the model in the multi-dimensional curve of the loss function could be in a local minimum. This could possibly make it difficult for the model to reach the minimum for the gold standard dataset.

- The gold standard contains about 10K images. Compared to the size of the arXiv dataset and based on the architecture of the Deepfigures model, this might not be sufficient to learn the task completely.

Figure 4.3: The plot of test accuracy of the Deepfigures model vs. the training step when its weights are initialized using the original Deepfigures model [54] and when trained on the gold standard dataset.

4.6 Experiment 6

4.6.1 Experimental setup

In Section 4.5 we use the original Deepfigures model for initializing the weights. However, as can be observed from Figure 4.3, the model performance is worsening. Therefore, in this experiment we try to leverage the features of the trained model weights of Deepfigures.

In transfer learning, after initializing the weights from a trained model, often the weights of some layers are frozen and only a few layers are trained. The reasoning behind this is that the feature maps that were learned in the trained model do not need to be trained again if the new task is in a similar domain.

Therefore, after initializing the weights with the original Deepfigures model, we freeze the weights of the ResNet-101 layers. During our training, we only train the remaining Overfeat layers towards the end of the model. The rest of the training process is identical to Section 4.5.

One of the advantages of this method is that this usually requires less data for model convergence since only the last few layers are training.

4.6.2 Results

Figure 4.4 shows the plot of the smoothed test accuracy against the training steps. We observe that the test accuracy decreases as the model training progresses. This means that the model's performance is worsening. Contrary to remarks made in Section 4.5.2, although we are training only the last few layers, the performance of the model does not improve. This answers the RQ4 posed in Section 1.2 that training the final layers of the Deepfigures

Figure 4.4: The plot of test accuracy of the Deepfigures model vs. the training step when its weights are initialized using the original Deepfigures model [54] and when only the final fully connected layers are trained on the gold standard dataset.

model (i.e. transfer learning) using the gold standard dataset does not improve the model performance.

4.7 Experiment 7

4.7.1 Experimental setup

In all of the previous experiments, we use the Deepfigures model. However, the best F1-score we are able to obtain yet is not more than 0.588. Therefore, we decide to experiment with a different object detection model. To this effect, we train the YOLOv5 model.

As mentioned in Section 2.1.4, we choose the extra large version of YOLOv5 with about 89 million trainable parameters. We train on the gold standard dataset and use 8-fold cross validation to report its performance. We use a batch size of 8.

4.7.2 Results

The results of the 8 fold cross validation are in Table 4.5. We observe that the mean F1-score of all the cross validation folds is 0.860. This F1-score is significantly higher that the original F1-score of Deepfigures. Further, this F1-score is also significantly higher than the F1-score obtained in any of our previous experiments.

Although the mean F1-score of 0.860 is significantly higher than the previous experiments, it is important to look at the standard deviation which tells us how consistently we are able to achieve this score. According to Table 4.5, the standard deviation of the F1-score is 0.0732. This means that the F1-score values do not vary more than 7.32% of the mean.

Further, in the previous experiments, we used the Deepfigures model architecture which uses a combination of ResNet-101 and Overfeat. The total number of trainable parameters in the Deepfigures model is about 45 million. However, YOLOv5 uses about 89 million trainable parameters. This means that the number of trainable parameters in YOLOv5 is almost double than that for Deepfigures. This could be one possible explanation for the higher F1-score achieved by YOLOv5 as compared to Deepfigures.

4.8 Discussion of results

In this section, we summarize and discuss the results we obtained in the experiments that we conducted.

Table 4.5: Performance of the original Deepfigures model on the gold standard dataset compared with the 8-fold cross validation of YOLOv5. (TPs=True Positives, FPs=False Positives, FNs=False Negatives)

Model	TPs	FPs	FNs	Precision	Recall	F1
Deepfigures	1005	1227	1143	0.450	0.468	0.459
YOLOv5 (K=0)	298	100	45	0.749	0.869	0.804
YOLOv5 (K=1)	262	39	57	0.870	0.821	0.845
YOLOv5 (K=2)	381	127	170	0.75	0.691	0.720
YOLOv5 (K=3)	282	22	8	0.928	0.972	0.949
YOLOv5 (K=4)	358	46	24	0.886	0.937	0.911
YOLOv5 (K=5)	457	58	32	0.887	0.935	0.910
YOLOv5 (K=6)	209	51	26	0.804	0.889	0.844
YOLOv5 (K=7)	261	43	19	0.859	0.932	0.894
YOLOv5 (Mean)	313.5	60.75	47.625	0.842	0.881	0.860
YOLOv5 (Std. dev.)	79.9	34.9	51.7	0.066	0.0899	0.0732

In Section 4.1, we described the POC we did to determine the feasibility of using data augmentation. The main goal of this experiment was to show whether using data augmentation on the born-digital training data (i.e., making the training data look more like scanned ETDs) helps improve the performance of figure extraction on scanned ETDs. From Table 4.1 we observed that using data augmentation indeed helps improve the performance. Our intuition behind this is that the selected data augmentation techniques brought the visual appearance of the born-digital ETDs closer to the scanned ETDs, thereby improving the performance of the model on scanned ETDs. However, in this experiment, we evaluated using only one ETD and did not have well-defined evaluation metrics.

Therefore, we conducted the next experiment (Section 4.2) to address these issues. We used well-defined evaluation metrics such as the IOU, TPs, FP, FN, Precision, Recall and the F1-score. We also used the gold standard dataset for stronger evaluation of the model. In Table 4.2, we report the F1-score of the original Deepfigures model by evaluating it on our gold standard dataset. Therefore, we are able to answer RQ1 as follows:

Answer to RQ1: The performance of the original Deepfigures is significantly lower for figure extraction from scanned ETDs as compared to its performance for figure extraction from born-digital documents. For figure extraction from scanned ETDs, it achieves an F1-score of 0.459.

In Section 4.3 and Section 4.4, we describe ablation studies in an attempt to find out which data augmentation transformations are helpful. For each of the models trained as part of these ablation studies, we chose the set of weights (a.k.a., checkpoints) which scored the highest value of F1-score on the validation set of the gold standard, and then reported the performance of the selected checkpoint on the test set of the gold standard. From the results obtained in Table 4.3 and Table 4.4, we did not observe any concrete trend in the data to prove the usefulness of the ablation studies in determining which transforms are helpful. However, since most of the model in the ablation studies achieved a significantly higher score than the original Deepfigures model, we were able to answer RQ2 as follows:

Answer to RQ2: The performance of the original Deepfigures model can be improved by training it on augmented data and by using weight initialization from the pre-trained model.

Further, as discussed in Section 4.5 and Section 4.6, we attempted to further improve the performance by training on the gold standard and by using transfer learning techniques, respectively. Specifically, as discussed in Section 4.5, we initialized the weights of the Deepfigures model and only trained the Overfeat layers. However, in both of these experiments, the performance of the model did not improve. Therefore, we can answer RQ3 and RQ4 as follows:

Answer to RQ3: The performance of the original Deepfigures model was not improved by training on manually labelled data.

Answer to RQ4: The performance of the original Deepfigures model was not improved by

using transfer learning techniques.

Finally, as in Section 4.7, we decide to use a different model since the highest F1-score we could achieve in the previous experiments was not significantly higher than the F1-score of the original Deepfigures model. Therefore, we train a YOLOv5 model on the gold standard dataset with 8-fold cross validation, a batch size of 8 and random weight initialization. The mean F1-score of the 8 cross validation folds was 0.860 with a standard deviation of 0.0732. We speculate that this significant boost in the F1-score as compared to any of the Deepfigures models in our previous experiments could be due to the difference in the number of trainable parameters in the two models. In Deepfigures, there are a total of about 45 million trainable parameters. However, in YOLOv5, the total number of trainable parameters is 89 million, which is almost double. This could be one explanation for the higher performance of YOLOv5. Further, it is also possible that the loss functions of the Deepfigures model could be stuck in a local minimum because of the weight initialization. This could make it difficult for it to come out of the local minimum and therefore prevent it from achieve it's global minimum on the augmented data.

Chapter 5

Future work

This research was conducted on a fairly large amount of data. The hyper-parameters we used were mostly the same as the ones used in Deepfigures [54]. However, since the task is now to extract figures from scanned ETDs, tuning the hyper-parameters might have improved the performance further; especially using a lower learning rate.

Since the ETDs sampled in the gold standard dataset span across multiple years, it would be interesting to evaluate the performance of our methods with respect to the dates on which the ETDs were published.

Another interesting analysis would be to find the performance of our methods separately on figures and tables. However, this would be possible only when there are separate training labels for figures and tables.

Further, the transformations tried out during the ablation studies were limited and chosen heuristically. A better approach would be to design a system to choose the transformations automatically. For example, some kind of function to measure the 'visual' similarity between two images of ETDs could help us choose transformations more effectively. This function to measure the similarity index could possibly be approximated by a machine learning model. However, until we come up with such a function to measure the similarity, trying out more combinations of ablation studies could help us reach the optimal combination of transforms.

It would also be interesting to report the performance of YOLOv5 after training it on the

arXiv dataset after applying the augmentations proposed in Sections 3.3.5 and 3.3.4.

Another interesting way to make born-digital ETDs look like scanned ETDs could leverage the recent advances in style transfer using CycleGANs. Research has been done to make any painting or image look like it was painted by, for example, Van Gogh [59].

Lastly, the model used for Deepfigures [54] is a combination of ResNet-101 and Overfeat. Several newer object detection models such as Single Shot Detector, Faster R-CNN, etc. have been proposed since. Using these could yield better results.

Chapter 6

Conclusions

This research focuses on extracting figures from scanned Electronic Theses and Dissertations (ETDs). We begin by describing the research problem, formulating the research questions that we are trying to answer, and reviewing the various works from the literature relevant to this work, such as Deepfigures and PDFFigures2.

We also propose LaTeX-based and image-based data transformations to augment the training data and thereby bring the feature distribution of the born-digital ETDs closer to that of scanned ETDs. To efficiently apply these proposed data transformations at scale, we discuss our system, which uses multiple parallel processes to pre-process the data for training.

All of the data we use was generated from born-digital ETDs. Previously, there was no ground-truth for evaluating the performance of our methods. Therefore, we curate a gold standard dataset consisting entirely of scanned ETDs and manually label it with bounding boxes around figures. This dataset consists of over 10k page images and 3.3k labels.

Finally, we describe the experiments we conducted to evaluate the performance of our proposed methods. In two of our experiments, we are able to obtain a set of model weights which are able to achieve an F1-score on the hidden (test) set greater than the original Deepfigures model. We also describe experiments in which we show that our approaches to training on the gold standard under the aforementioned experimental setup does not improve the performance, even after using transfer learning.

Appendices

Appendix A

Software Details

A.1 Technology stack

The technology stack used for this research is based on the Python 3 programming language. We also use Bash for scripting. Following are some of the tools we used in this work.

A.1.1 Anaconda 3

Anaconda is an enterprise-ready, secure, and scalable data science platform that empowers teams to govern data science assets, collaborate, and deploy data science projects [1]. However, we use it mainly as our Python environment manager. As a result, to run our code on a different computer, the software environment can be easily set up using Anaconda and the requirements file we provide in our source code.

A.1.2 Docker

Docker is a set of software tools to package your application to enable it to run in an isolated and platform-agnostic container [43]. We use it to package our code. This makes it simple to run our code on any computer which has a Docker process running without setting up the Anaconda environment.

A.1.3 TensorFlow

TensorFlow is a machine learning system that operates at large scale and in heterogeneous environments. TensorFlow uses data-flow graphs to represent computation, shared state, and the operations that mutate that state [20]. Our code uses TensorFlow as a part of the TensorBox framework [4] for implementing the machine learning models that we use.

A.1.4 PyTorch

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab [47]. Although we use TensorFlow for implementing the machine learning models used in this work, we use PyTorch for implementing our scalable data processing system proposed in Section 3.5.

A.1.5 TexLive

We use PDFLaTeX to compile the LaTeX source code of the arXiv papers. TexLive is a tool which contains PDFLaTeX and is intended to be a straightforward way to get up and running with the TeX document production system [11].

A.1.6 VGG Image Annotator

VIA is the acronym for VGG Image Annotator. We use VIA for manually drawing bounding boxes for our gold standard dataset. VIA was released by the Visual Geometry Group (VGG) at the University of Oxford [27, 28, 55].

A.2 Github repository and code organization

Our entire code is available at our Github repository [36]. This repository contains the *README* file which has the instructions to set up and run our code on any computer. In this section we describe the main modules and directories in which our code is organised.

A.2.1 ./bin

This directory contain the binaries for the PDFFigures *.jar* files. These binaries were used in the original code of Deepfigures [54] for comparing the results with PDFFigures.

A.2.2 ./deepfigures

This directory contains the code for the Deepfigures pipeline and our modified Deepfigures pipelines. It also contains the code for making the inference from the PDF files on a trained model.

A.2.3 ./dockerfiles

This directory contains the *Dockerfiles*. The *Dockerfiles* in this directory are used to package our code into Docker containers.

A.2.4 ./gold_standard

This directory contains the source code for curating the gold standard dataset. This includes scripts to crawl the MIT website, download the PDF and metadata files, sample from the

downloaded files, convert the PDF files to the list of images, visualize the annotations, convert the annotations to various common formats, etc.

A.2.5 ./hpc

We conduct most of our experiments on Virginia Tech's (VT) Advanced Research Computing (ARC) computer cluster. This directory contains all of the Bash scripts to launch and manage the jobs on ARC for our various experiments.

A.2.6 ./models

This directory contains the sample hyper-parameters for the machine learning models that we use.

A.2.7 ./my_tests

This directory contains a few unit tests.

A.2.8 ./scripts

A script called *manage.py* in the root directory of the source code is used to run a few frequently used operations, such as launching the training, pre-processing the data downloaded from arXiv, running inference on a given PDF, and so on. The *./scripts* directory contains the code for each of these operations which is invoked by the *manage.py* script.

A.2.9 ./vendor

This directory contains the source code from TensorBox framework [4] that we use to train the Deepfigures model.

A.2.10 ./yolov5

This directory contains the source code that is used in conjunction with the YOLOv5 source code [12].

Appendix B

User manual

B.1 Software environment setup

The code in our repository uses many software libraries. Therefore, to be able to run our code, these software libraries need to be set up on the computer on which we need to run it. Sections B.1.1 and B.1.2 describe the two ways of setting up the environment.

The methods described in Sections B.1.1 and B.1.2 need Nvidia's CUDA toolkit v9.0.176 and Nvidia's CUDNN library v7.1 installed for using the GPU. These come pre-installed in ARC. To use these two libraries, request a GPU compute node and execute the following two commands to load these modules:

```
module load cuda/9.0.176
module load cudnn/7.1
```

On any other system, Nvidia's official installation guides available on the internet can be used.

B.1.1 Docker

Before running the code using Docker, Docker should be installed and the Docker daemon should be running on the host system. To build a Docker image using the Dockerfiles from

the source code, run the following command at the root of the source code:

```
# Create and set up the Anaconda environment
ENV_NAME=deepfigures_3
conda remove --name $ENV_NAME --all -y
conda create --name $ENV_NAME python=3.6 -y
source activate $ENV_NAME
pip install -r requirements.txt --no-cache-dir

# Build the Docker images from the Dockerfiles.
python manage.py build
```

This will build two Docker images, one image that uses the GPU and another that only uses the CPU. (Note: Although we are using Docker, the Anaconda environment still needs to be set up to install the software libraries used in the manage.py script in the above set of commands.) Once the Docker images are built, run the desired image. In the following command, we run the GPU image:

```
docker run -it sampyash/vt_cs_6604_digital_libraries:deepfigures_gpu_0.0.6
```

The above command will run the recently built Docker image. Remember to change the image version to the one that you built if needed. This command will give a Bash shell session inside the Docker image. Any further commands, such as training the model, generating the data, or running inference can be easily run inside this Docker session using manage.py.

Pre-built Docker images can also be found in our Docker hub repository at: https://hub.docker.com/r/sampyash/vt_cs_6604_digital_libraries/tags. To run these pre-built Docker images, just the above 'docker run' command is sufficient. However, building the

Docker image from the source as mentioned above will give the latest version of the source code.

When Docker is not installed on the host

When Docker is not installed and running on the host system, it is recommended to install and start it. However, to install and start Docker, root user privileges are required, which are not always available. When root privileges are not available and a Docker image needs to be run, it is possible to use Singularity instead. Singularity is an alternative to Docker for HPC systems. It is containerization software similar to Docker and is used to package your apps.

Further, the Docker daemon needs to run with root level privileges. This Docker daemon is responsible for running all the Docker images on the system. Therefore, any Docker that is run, runs with root level privileges irrespective of whether that Docker image needs the privileges or not. This essentially gives the user root level privileges on the host system, which is often not desired on multi-tenant systems such as an HPC cluster. Singularity does not run its images with root level privileges, and hence is a suitable alternative to Docker on HPC systems. Further, Singularity is also able to run pre-built Docker images. However, any functions in that Docker image which need root level privileges might not run properly. None of the Docker containers in our work need root level privileges, and hence can be safely run using Singularity.

If Singularity is installed on the host system, pre-built Docker images from Docker hub can be directly run as follows:

```
# This command will pull and convert the Docker image into a .sif file
# and store it in the configured Singularity directory on the host.
```

```
# Note: If you want to build a fresh image from source,
# you can build a new image and push it to Docker Hub.
singularity pull \
    docker://sampyash/vt_cs_6604_digital_libraries:deepfigures_gpu_0.0.6

# Edit this with your Singularity directory.
SINGULARITY_DIR=/work/cascades/sampanna/singularity

# Run the pulled image.
singularity run --nv \
    -B /home/sampanna/deepfigures-results:/work/host-output \
    -B /home/sampanna/deepfigures-results:/work/host-input \
    $SINGULARITY_DIR/vt_cs_6604_digital_libraries_deepfigures_cpu_0.0.6.sif \
    /bin/bash
```

The above commands will pull the *sampyash/vt_cs_6604_digital_libraries:deepfigures_gpu_0.0.6* Docker image from Docker Hub, convert it into a *.sif* file, and return a shell session inside the pulled container. This shell session then can be used to run any of the scripts from the source code.

B.1.2 Anaconda

Another way of running the code is to set up the Anaconda environment and run the code natively on the operating system, without Docker. We do this by installing all the required dependencies inside an Anaconda environment using the following commands:

```
# Create and set up the Anaconda environment
```

```
ENV_NAME=deepfigures_3
conda remove --name $ENV_NAME --all -y
conda create --name $ENV_NAME python=3.6 -y
source activate $ENV_NAME
pip install -r requirements.txt --no-cache-dir
```

Further, the Deepfigures and the TensorBox modules need to be installed in the current environment using their respective setup scripts.

```
# Edit this to point to the location where you cloned the source code.
PROJECT_SOURCE=/home/sampanna/deepfigures-open

# This will remove any previously generated build files.
# If none are present, it won't do anything.
rm -rf $PROJECT_SOURCE/build \
  $PROJECT_SOURCE/dist \
  $PROJECT_SOURCE/deepfigures_open.egg-info \
  $PROJECT_SOURCE/vendor/tensorboxresnet/.eggs \
  $PROJECT_SOURCE/vendor/tensorboxresnet/build \
  $PROJECT_SOURCE/vendor/tensorboxresnet/dist \
  $PROJECT_SOURCE/vendor/tensorboxresnet/tensorboxresnet.egg-info \
  $PROJECT_SOURCE/vendor/tensorboxresnet/tensorboxresnet/utils/stitch_wrapper.cpp

# Uninstall previously installed deepfigures-open and tensorboxresnet, if any.
pip uninstall deepfigures-open -y
pip uninstall deepfigures-open -y
```

```
pip uninstall tensorboxresnet -y
pip uninstall tensorboxresnet -y

cd $PROJECT_SOURCE

# Install deepfigures-open in the current environment.
python setup.py install

# Install tensorboxresnet in the current environment.
cd vendor/tensorboxresnet && python setup.py install && cd ../..

# Remove any generated build files to keep the source code clean.
rm -rf $PROJECT_SOURCE/build \
  $PROJECT_SOURCE/dist \
  $PROJECT_SOURCE/deepfigures_open.egg-info \
  $PROJECT_SOURCE/vendor/tensorboxresnet/.eggs \
  $PROJECT_SOURCE/vendor/tensorboxresnet/build \
  $PROJECT_SOURCE/vendor/tensorboxresnet/dist \
  $PROJECT_SOURCE/vendor/tensorboxresnet/tensorboxresnet.egg-info \
  $PROJECT_SOURCE/vendor/tensorboxresnet/tensorboxresnet/utils/stitch_wrapper.cpp
```

After executing the above commands, the Anaconda environment should be ready for use.

Amazon Web Services (AWS) integration

Optionally, to be able to download the arXiv papers from AWS cloud buckets as mentioned in [58], AWS credentials need to be added to the *credentials* file present in the root directory

of the source code. This needs to be done because the arXiv data which is present in AWS buckets is in *requester-pays* type of buckets. This means that one needs to pay to download this data.

A sample of the *credentials* file looks as follows:

```
[default]
aws_access_key_id=dummy_key_id
aws_secret_access_key=dummy_secret_access_key
aws_session_token=dummy_session_token
```

Replace the values in this file with the appropriate credentials from your AWS account.

In case your AWS account does not have the above keys, you can also set your credentials in the *deepfigures-local.env* file present in the root directory of the source code.

TexLive setup

Optionally, to be able to process the arXiv data downloaded from AWS and to generate the labels for training, TexLive needs to be set up. Using the Quick Install instructions at https://www.tug.org/texlive/quickinstall.html, TexLive can be set up on your system. To install TexLive on a system without root level privileges, install it in a directory for which you have sufficient privileges (e.g., your home directory). Depending on the system's performance and the download speed, the entire installation process can take about an hour. The total disk space consumed by the full installation is about 7 GB.

B.2 Recommended hardware

We use the ARC cluster extensively for running our code. ARC contains many compute nodes with different hardware configurations. The nodes we most frequently used in this work are the ones which have GPU. Typically, the compute nodes we used had about 32 cores and 2 Nvidia P100 GPUs attached. These compute nodes had about 512 GB of memory and each of the P100 GPUs had about 12 GB of memory.

For tasks which do not require hardware accelerators, such as running inference on the models, generating the labels from the arXiv dataset, downloading the arXiv dataset, etc., GPUs are not needed. Therefore, similar compute nodes without GPUs can be used. However, for training the machine learning models, using GPUs is highly recommended.

Further, for each user, ARC provides about 6 TB of high throughput disk storage which is useful for overcoming any data loading bottlenecks while training the machine learning models.

B.3 Reproducing the experiments

In Chapter 4, we described the various experiments that we conducted. All of our experiments were run on ARC. To run an experiment on ARC, a shell script is needed which describes the requested hardware resources needed to run the job and the exact modules, environment, and the commands that need to be executed to run the experiment. All such scripts that we used in this work are available in the *./hpc* directory as described in Section A.2.5.

Following is an example of one such script which launches the training of the Deepfigures model. The comments in the following scripts above each group of lines describe what that

group of lines does.

```
#!/bin/bash -x

## Resource requests if running on newriver.
#PBS -l nodes=1:ppn=28:gpus=1
#PBS -l walltime=20:00:00
#PBS -q p100_normal_q
#PBS -A your_allocation_name
#PBS -W group_list=newriver
#PBS -M youremail@vt.edu
#PBS -m bea

## Resource requests if running on cascades.
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --gres=gpu:1
#SBATCH --mail-type=ALL
#SBATCH --mail-user=youremail@vt.edu
#SBATCH -t 20:00:00
#SBATCH -p v100_normal_q
#SBATCH -A your_allocation_name

# Experiment name for pre-fixing checkpoint directory names.
EXPERIMENT_NAME=377268_arxiv_eval_included_checkpoint_lr_test
```

```
# Defining the timestamp for naming directories.
current_timestamp() {
  date +"%Y-%m-%d_%H-%M-%S"
}
ts=$(current_timestamp)

# Loading modules.
module purge
module load cuda/9.0.176
module load cudnn/7.1

# Setting up the environment.
CONDA_ENV=deepfigures_3
PYTHON=/home/sampanna/.conda/envs/"$CONDA_ENV"/bin/python

# System-specific setup commands.
if [ "$SYSNAME" = "cascades" ]; then
  # Commands to be executed if the host system is cascades.
  module load Anaconda/5.1.0
  module load gcc/7.3.0
  source activate "$CONDA_ENV"
  SCRATCH_DIR=$TMPRAM # 311 GB on v100 nodes. I/O: 331 MBPS.
elif [ "$SYSNAME" = "newriver" ]; then
  # Commands to be executed if the host system is cascades.
  module load Anaconda/5.2.0
  module load gcc/6.1a.0
```

```
  source activate "$CONDA_ENV"
  SCRATCH_DIR=$TMPFS # 429 GB on p100 nodes. I/O: 770 MBPS.
else
  echo "Unrecognised system. Exiting."
  exit 1
fi

# Defining environment variables.
DEEPFIGURES_RESULTS=/work/cascades/sampanna/deepfigures-results
SOURCE_CODE=/home/sampanna/deepfigures-open
ZIP_DIR=$DEEPFIGURES_RESULTS/pregenerated_training_data/377268
WEIGHTS_PATH=$DEEPFIGURES_RESULTS/weights/save.ckpt-500000
HYPES_PATH=$SOURCE_CODE/models/sample_hypes.json
MAX_ITER=10000000
LOG_DIR=$DEEPFIGURES_RESULTS/model_checkpoints
DATASET_DIR=$DEEPFIGURES_RESULTS/arxiv_coco_dataset
TRAIN_IDL_PATH=$DATASET_DIR/figure_boundaries_train.json
TRAIN_IMAGES_DIR=$DATASET_DIR/images
TEST_IDL_PATH=$DATASET_DIR/figure_boundaries_test.json
TEST_IMAGES_DIR=$DATASET_DIR/images
GOLD_STANDARD_DATASET_DIR=$DEEPFIGURES_RESULTS/gold_standard_dataset
HIDDEN_IDL_PATH=$GOLD_STANDARD_DATASET_DIR/figure_boundaries.json
HIDDEN_IMAGES_DIR=$GOLD_STANDARD_DATASET_DIR/images
MAX_CHECKPOINTS_TO_KEEP=200
TEST_SPLIT_PERCENT=20
USE_GLOBAL_STEP_FOR_LR=False
```

```
# Starting the training.
$PYTHON -m tensorboxresnet.train \
  --weights "$WEIGHTS_PATH" \
  --gpu="$CUDA_VISIBLE_DEVICES" \
  --hypes="$HYPES_PATH" \
  --max_iter="$MAX_ITER" \
  --logdir="$LOG_DIR" \
  --experiment_name="$EXPERIMENT_NAME" \
  --train_idl_path="$TRAIN_IDL_PATH" \
  --test_idl_path="$TEST_IDL_PATH" \
  --hidden_idl_path="$HIDDEN_IDL_PATH" \
  --train_images_dir="$TRAIN_IMAGES_DIR" \
  --test_images_dir="$TEST_IMAGES_DIR" \
  --hidden_images_dir="$HIDDEN_IMAGES_DIR" \
  --max_checkpoints_to_keep="$MAX_CHECKPOINTS_TO_KEEP" \
  --timestamp="$ts" \
  --scratch_dir="$SCRATCH_DIR" \
  --zip_dir="$ZIP_DIR" \
  --test_split_percent="$TEST_SPLIT_PERCENT" \
  --use_global_step_for_lr "$USE_GLOBAL_STEP_FOR_LR"

echo "Job ended. Job ID: $SLURM_JOBID"
exit 0
```

Further, all hyper-parameters required to train the model are defined in a separate *.json* file.

All such files used for our experiments are available in the *./hpc* directory.

These *.json* files also define the seeds of the random number generators in our code. This way, the experiments can be reproduced in a deterministic way.

Most of the hyper-parameters we use in our experiments are similar to the ones mentioned in the above code snippet. Any significant changes for running a certain experiment are mentioned in the experimental setup section of each experiment in Chapter 4.

Appendix C

Additional figures

C.1 Sample images from the gold standard dataset

Following are some randomly sampled images from the gold standard showing the manually annotated bounding boxes.

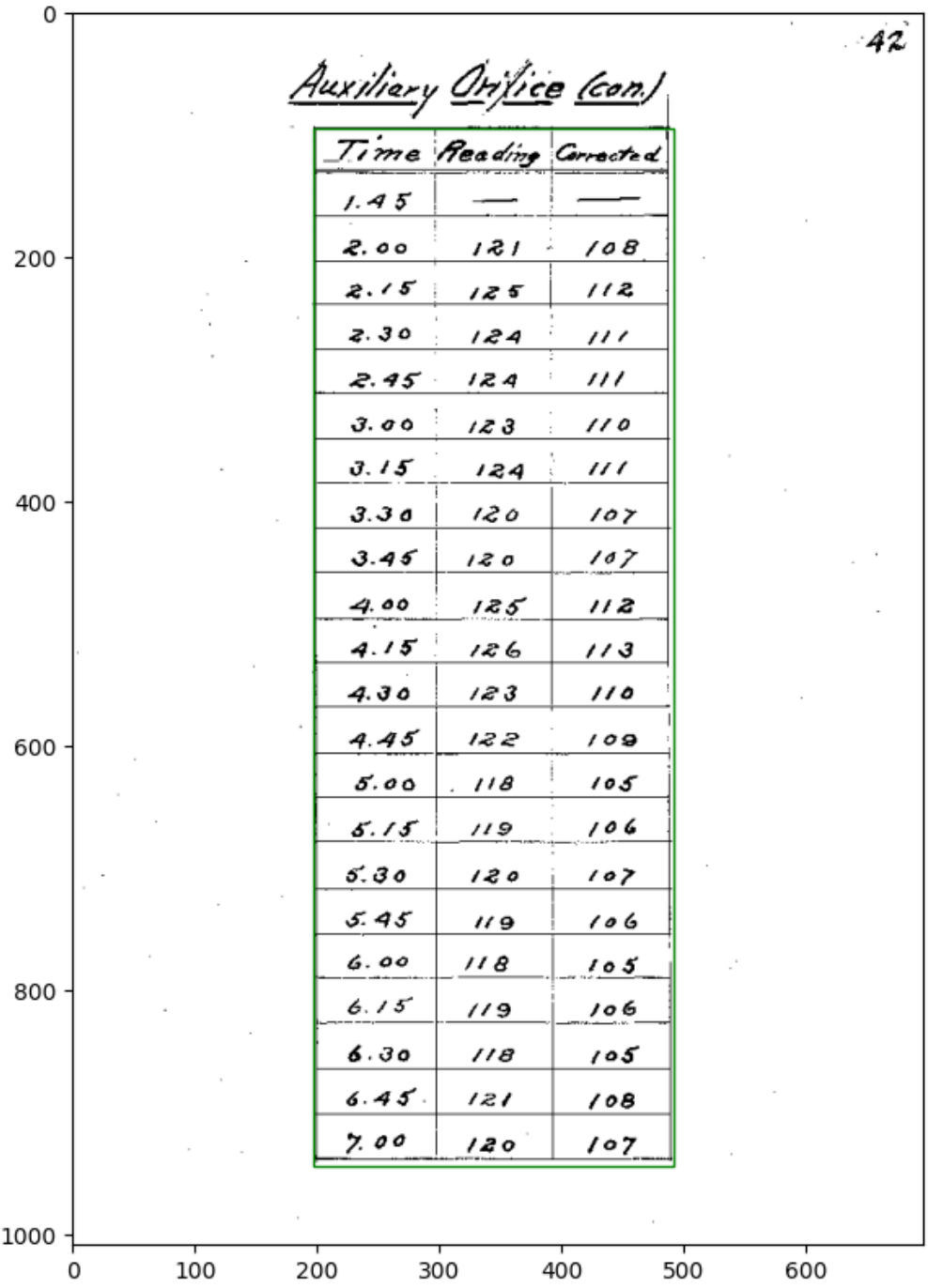


Figure C.1: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/100321, page number: 049.

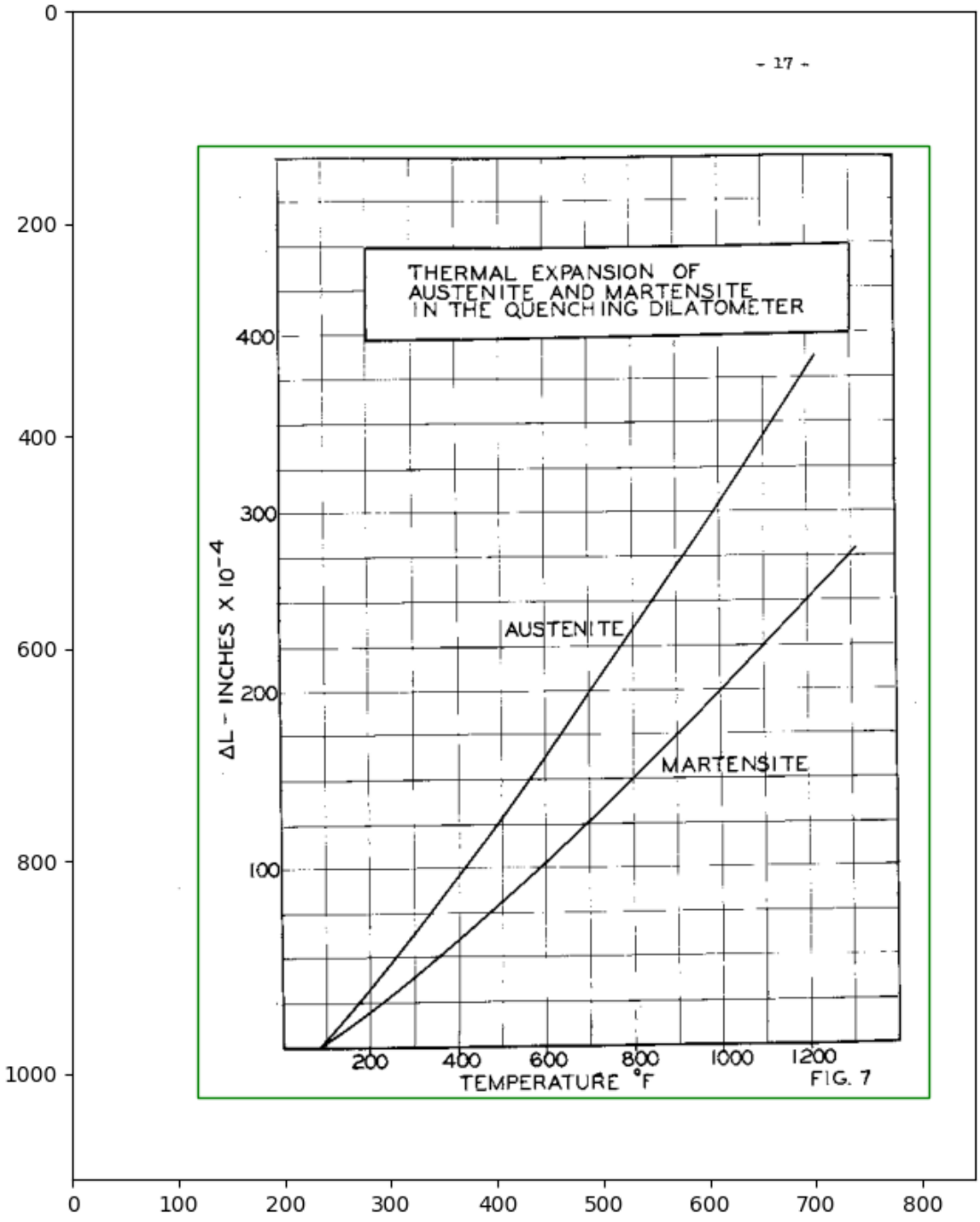
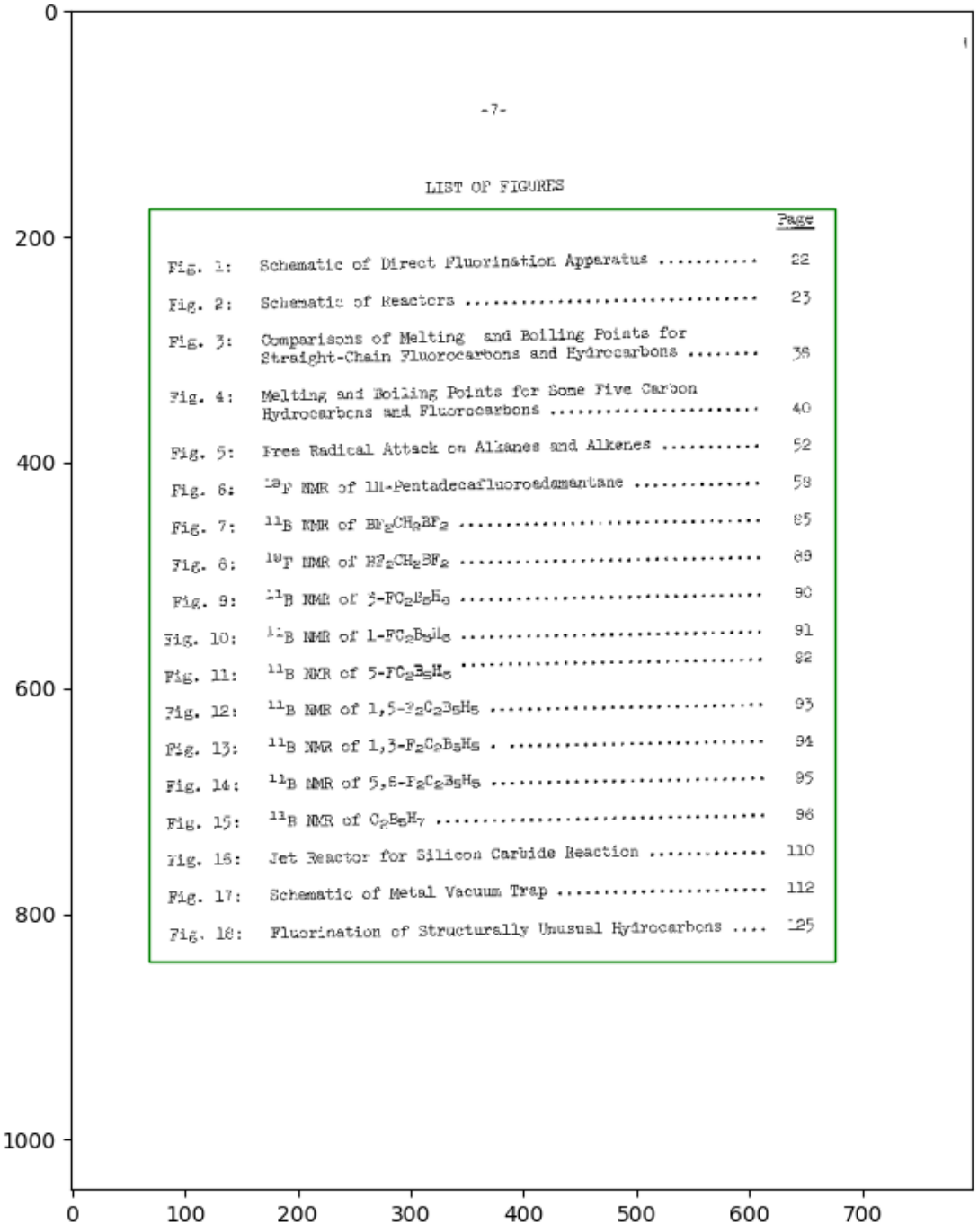


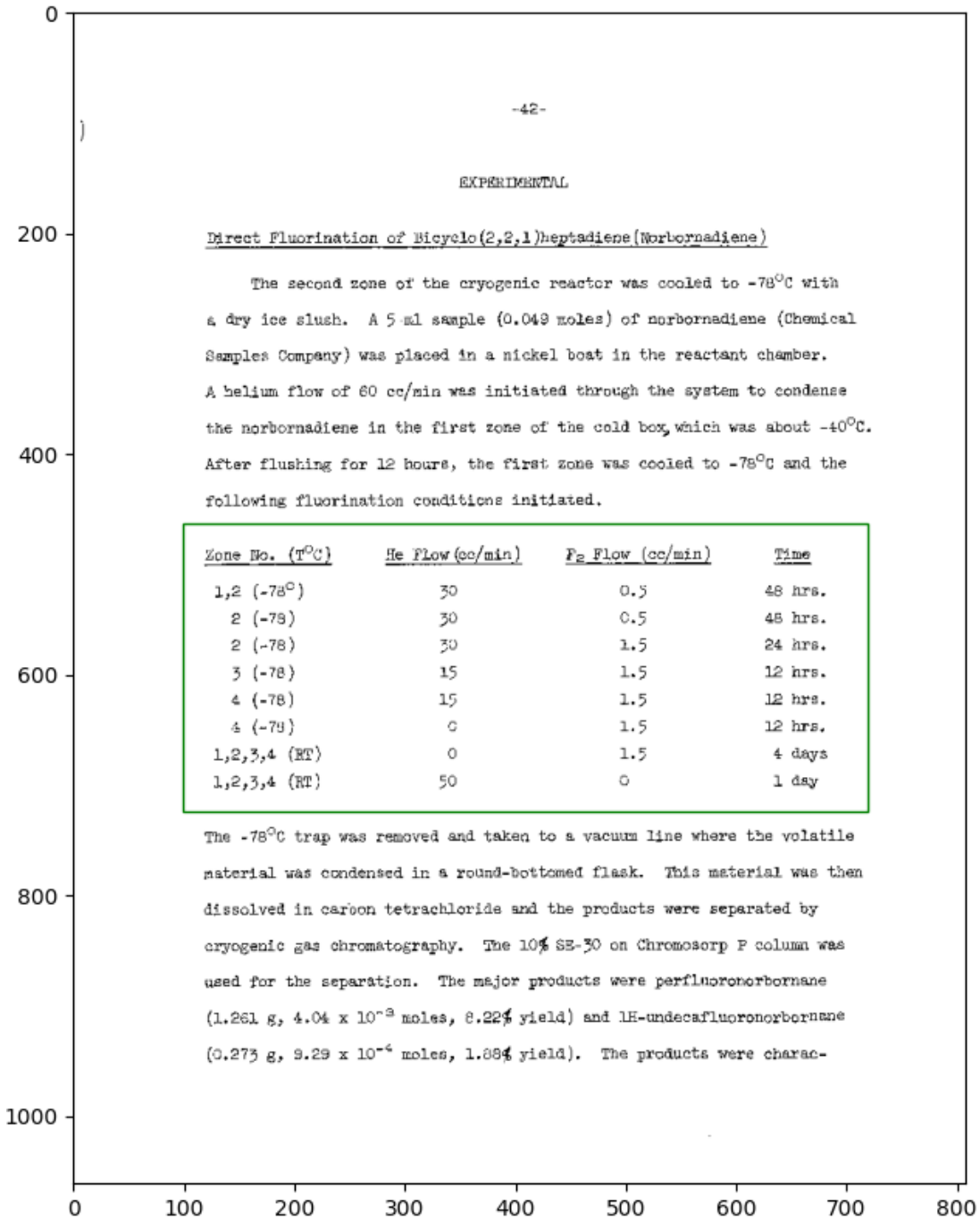
Figure C.2: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/103195, page number: 027.



-7-

LIST OF FIGURES

Figure C.3: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/13252, page number: 007.



-42-

EXPERIMENTAL

Direct Fluorination of Bicyclo(2,2,1)heptadiene(Norbornadiene)

The second zone of the cryogenic reactor was cooled to -78°C with a dry ice slush. A 5 ml sample (0.049 moles) of norbornadiene (Chemical Samples Company) was placed in a nickel boat in the reactant chamber. A helium flow of 60 cc/min was initiated through the system to condense the norbornadiene in the first zone of the cold box, which was about -40°C. After flushing for 12 hours, the first zone was cooled to -78°C and the following fluorination conditions initiated.

Zone No. (T°C)	He Flow (cc/min)	F_2 Flow (cc/min)	Time
1,2 (-78°)	30	0.5	48 hrs.
2 (-78)	30	0.5	48 hrs.
2 (-78)	30	1.5	24 hrs.
3 (-78)	15	1.5	12 hrs.
4 (-78)	15	1.5	12 hrs.
4 (-78)	0	1.5	12 hrs.
1,2,3,4 (RT)	0	1.5	4 days
1,2,3,4 (RT)	50	0	1 day

The -78°C trap was removed and taken to a vacuum line where the volatile material was condensed in a round-bottomed flask. This material was then dissolved in carbon tetrachloride and the products were separated by cryogenic gas chromatography. The 10% SE-30 on Chromosorp P column was used for the separation. The major products were perfluoronorbornane (1.261 g, 4.04×10^{-3} moles, 8.22% yield) and 1H-undecafluoronorbornane (0.273 g, 9.29×10^{-4} moles, 1.88% yield). The products were charac-

Figure C.4: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/13252, page number: 042.

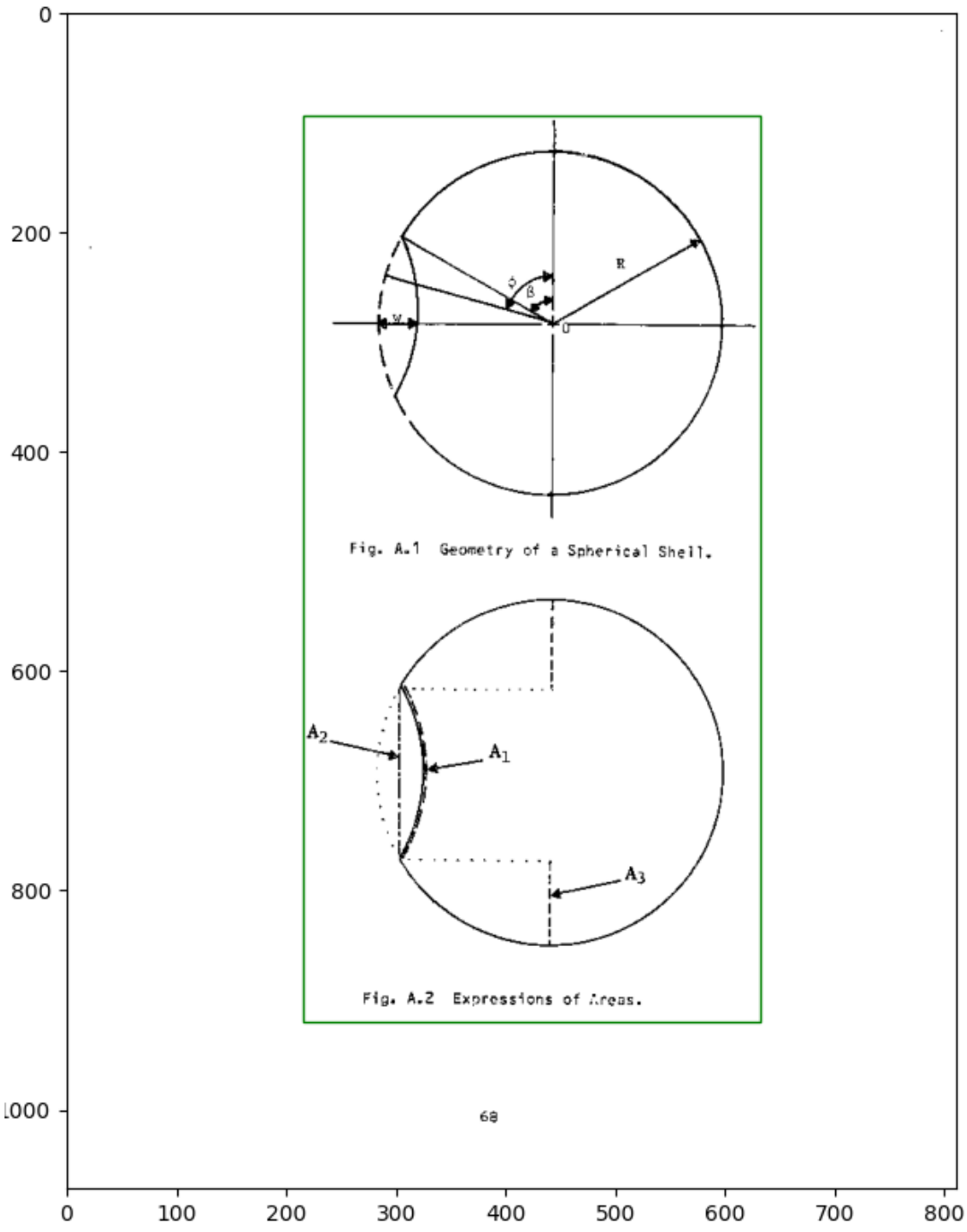


Figure C.5: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/15434, page number: 068.

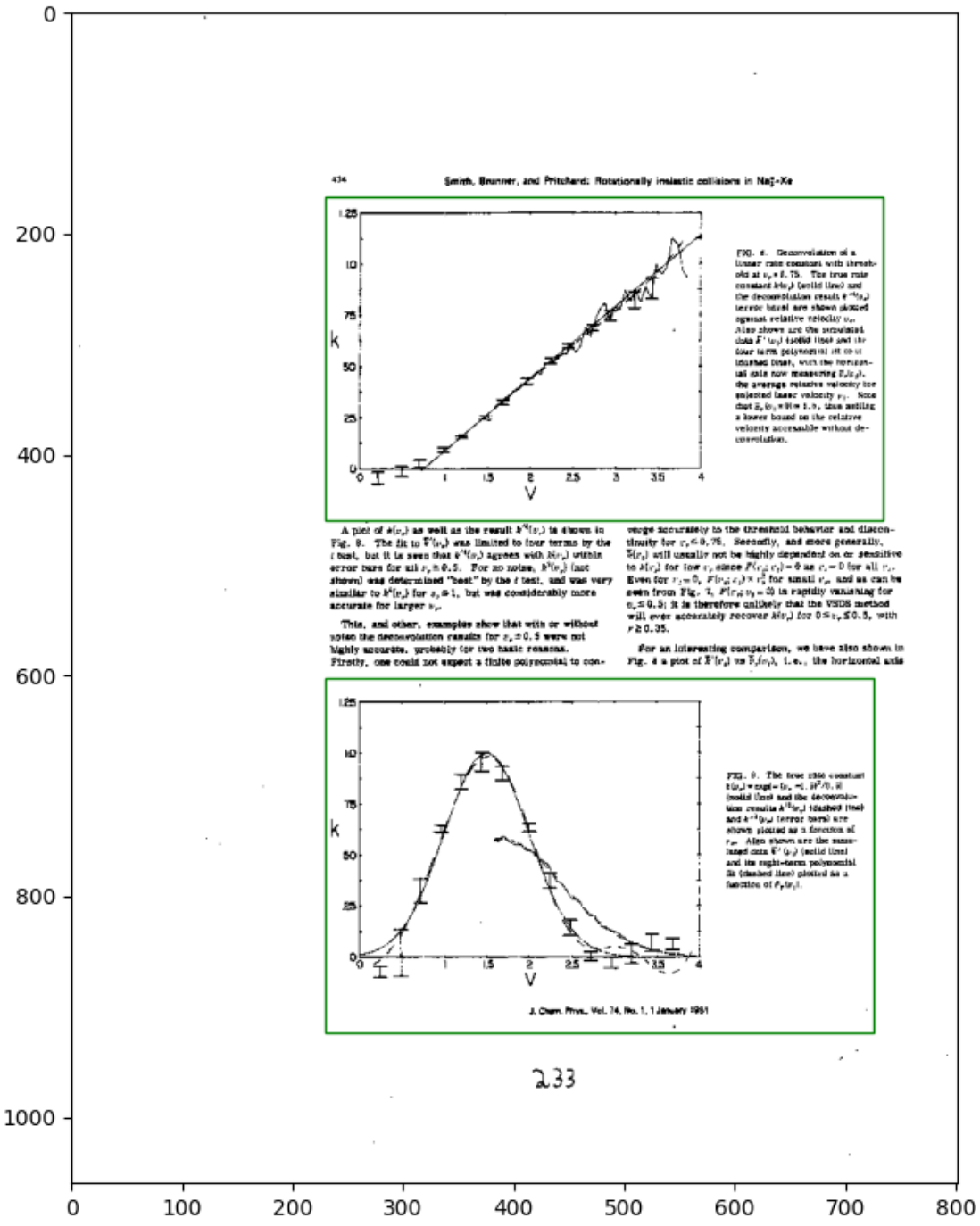


Figure C.6: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/15568, page number: 233.

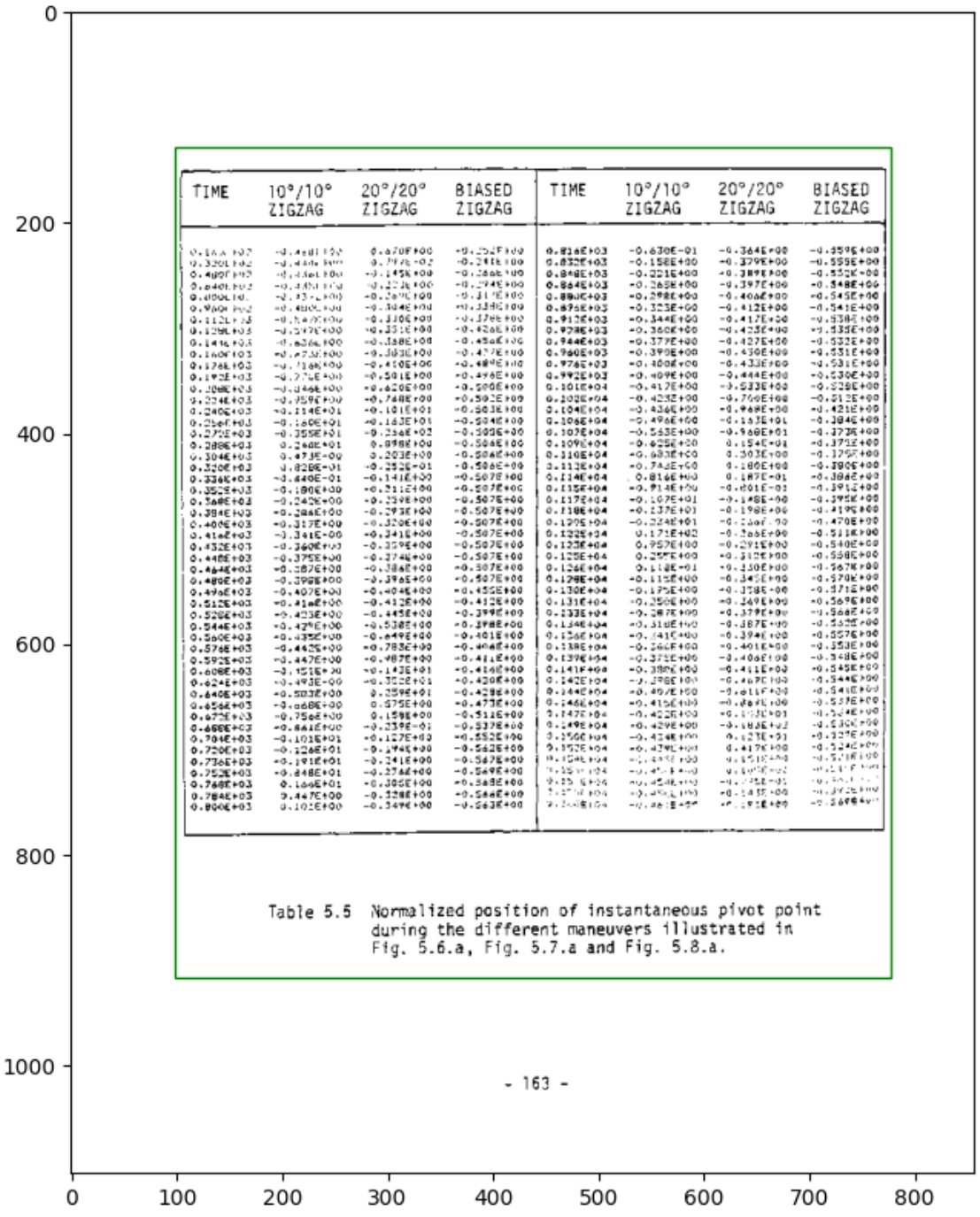


Figure C.7: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/15820, page number: 164.

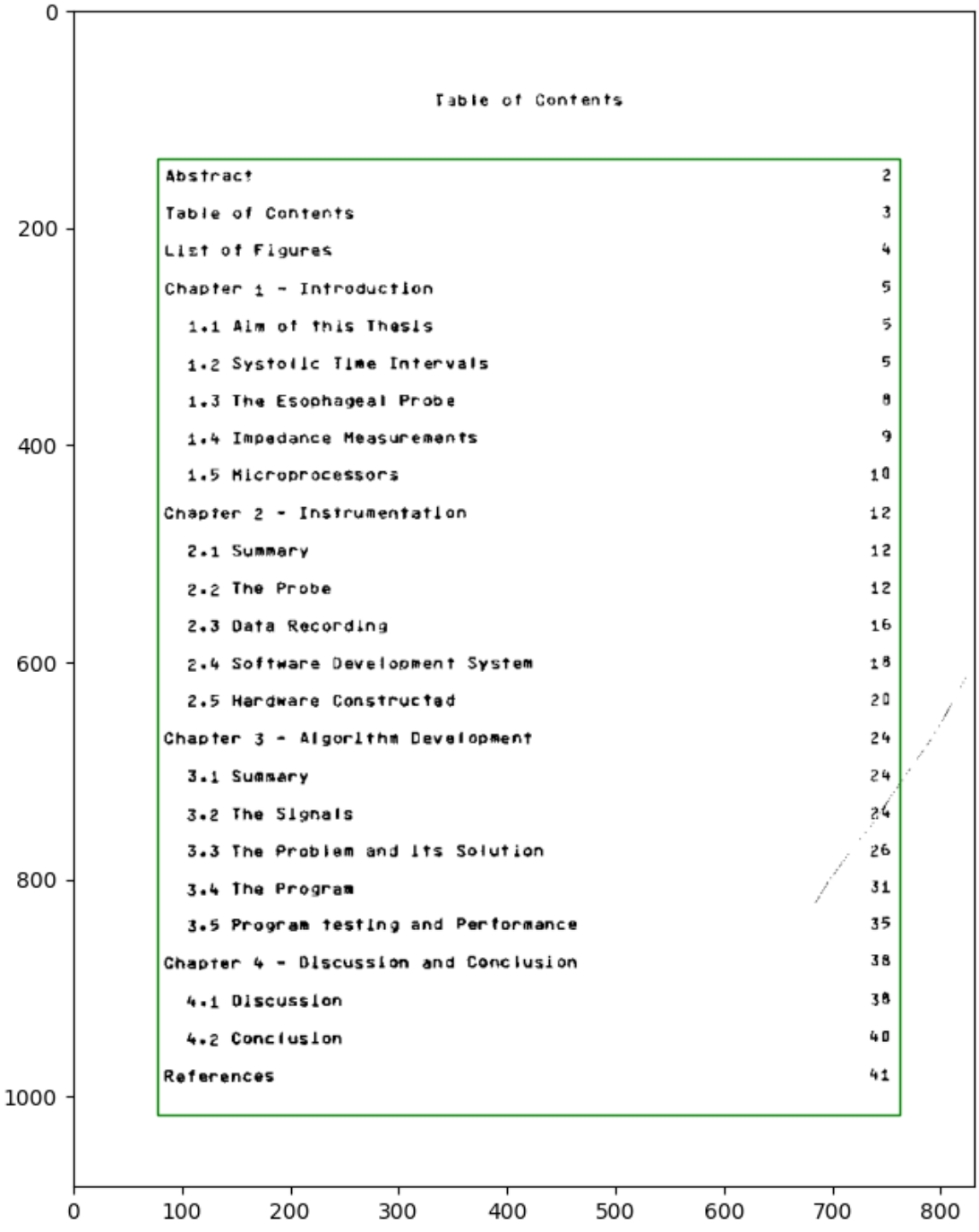


Figure C.8: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/16394, page number: 003.

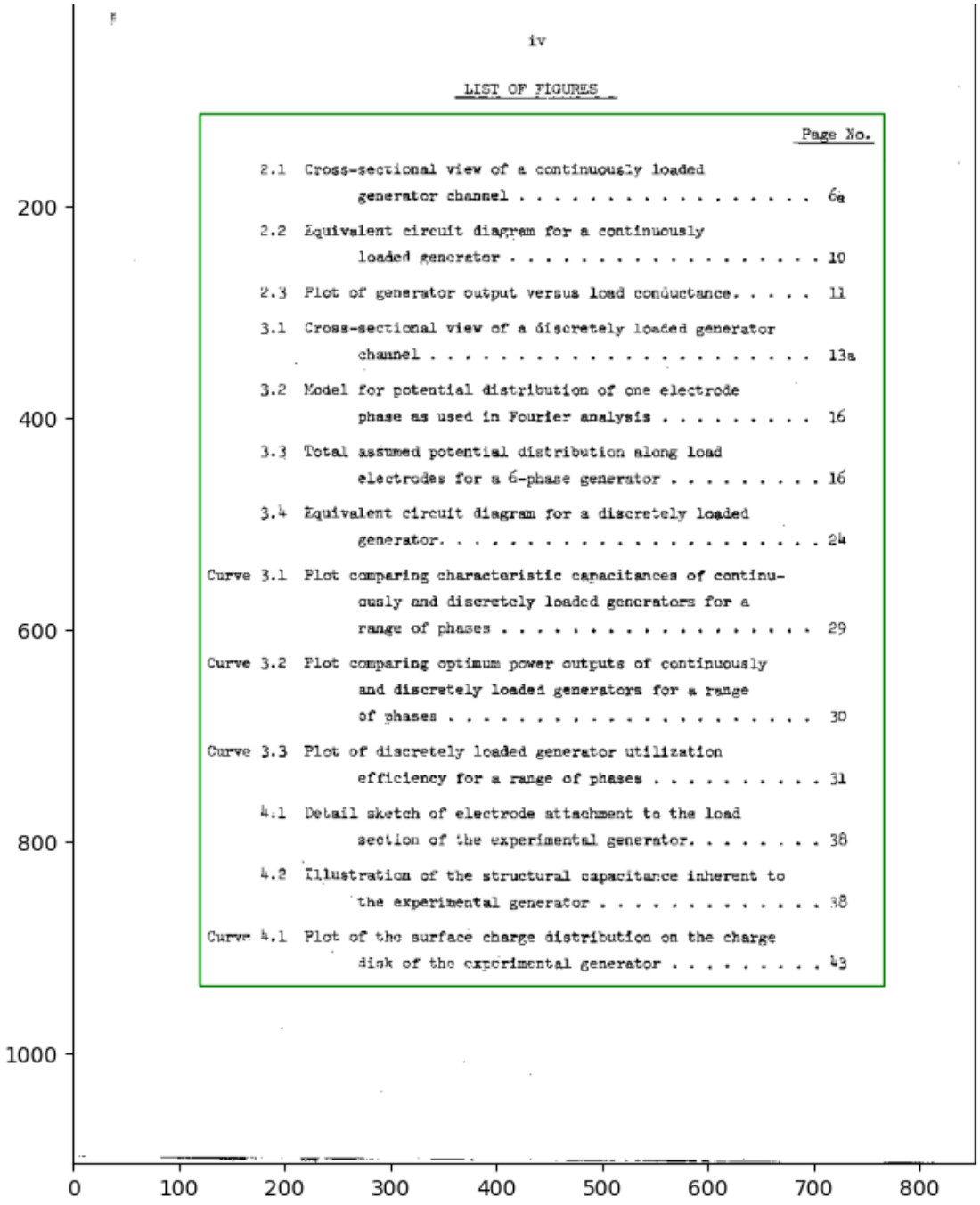


Figure C.9: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/17465, page number: 004.

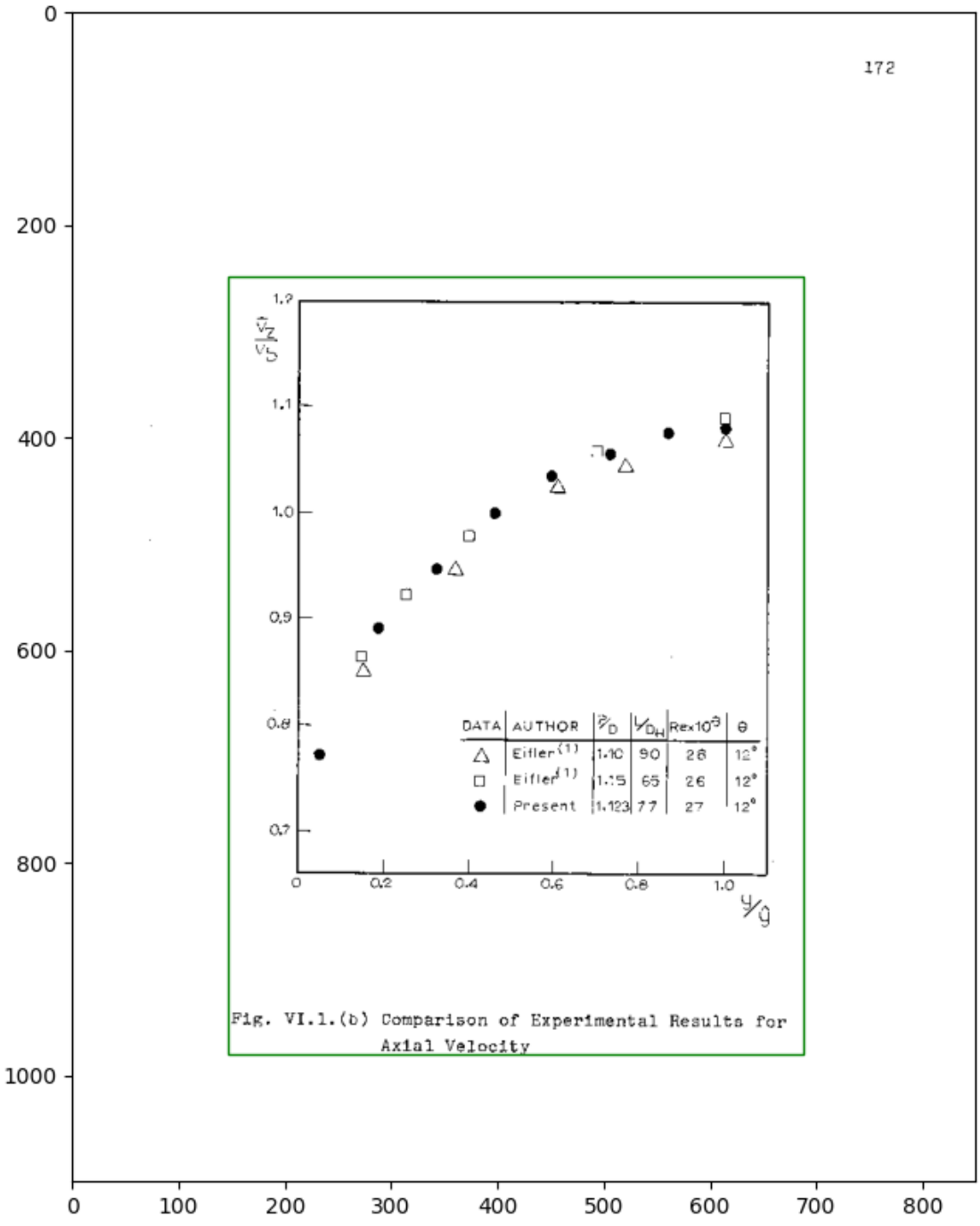


Figure C.10: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/27399, page number: 172.

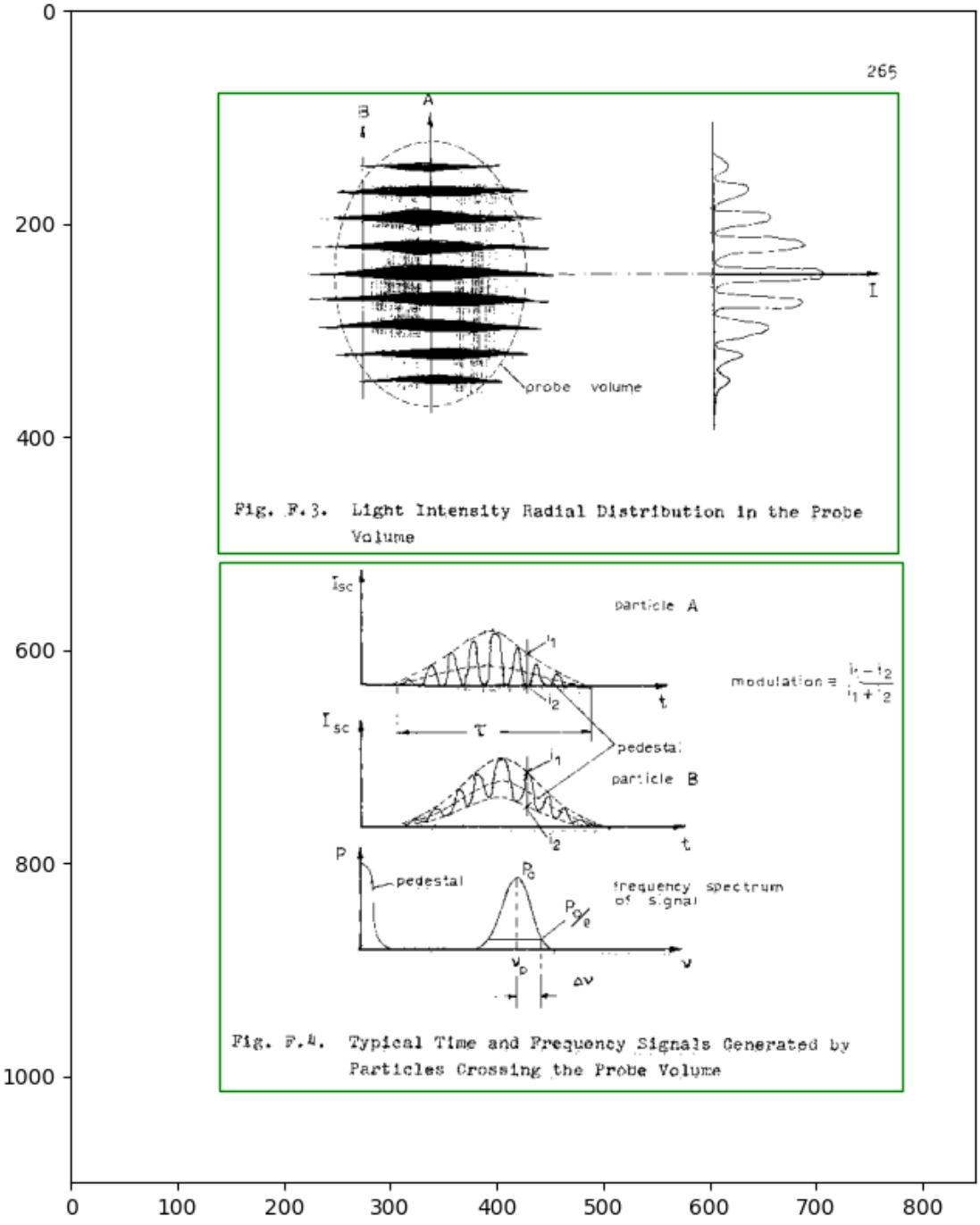


Figure C.11: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/27399, page number: 266.

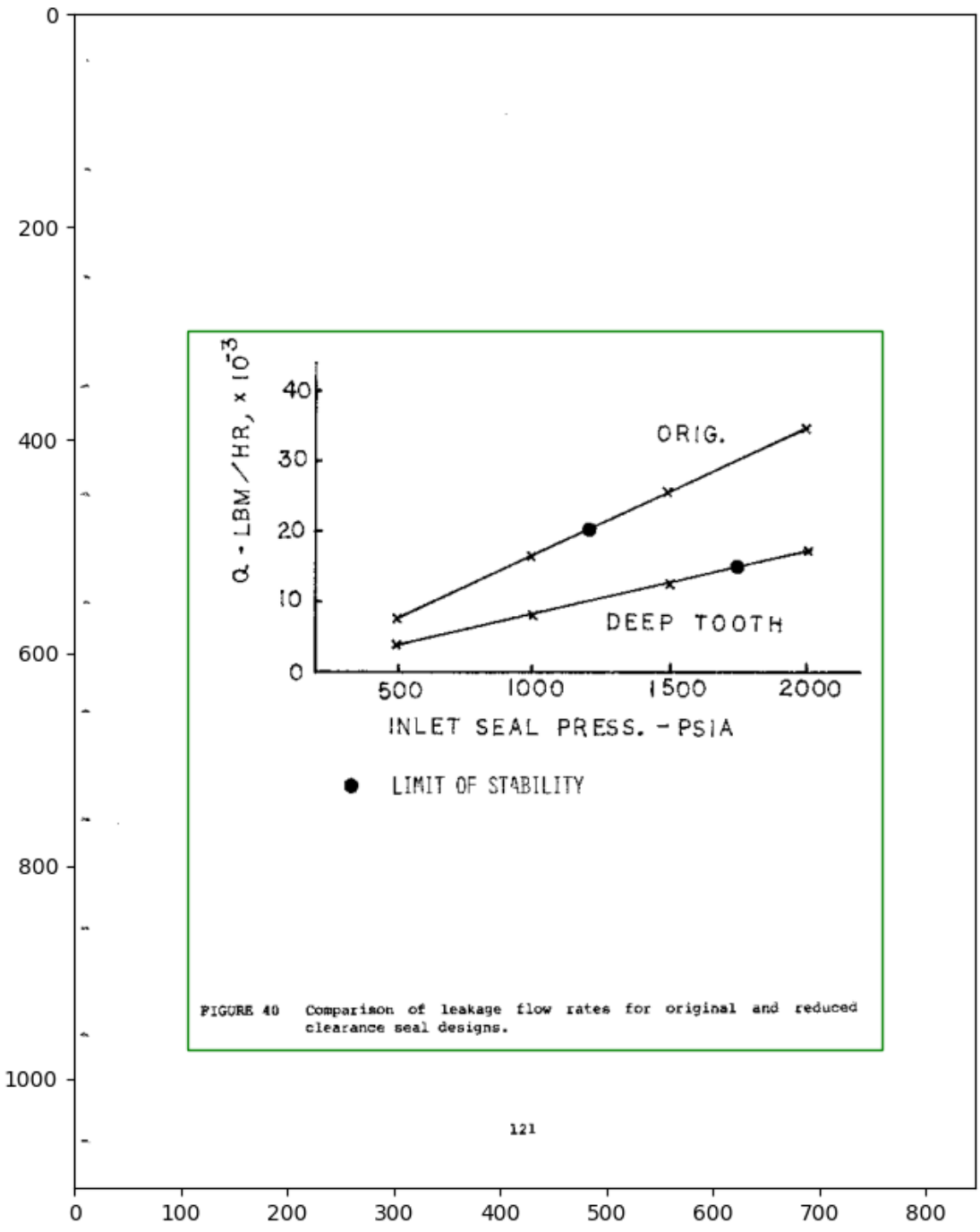


FIGURE 40 Comparison of leakage flow rates for original and reduced clearance seal designs.

Figure C.12: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/29848, page number: 133.

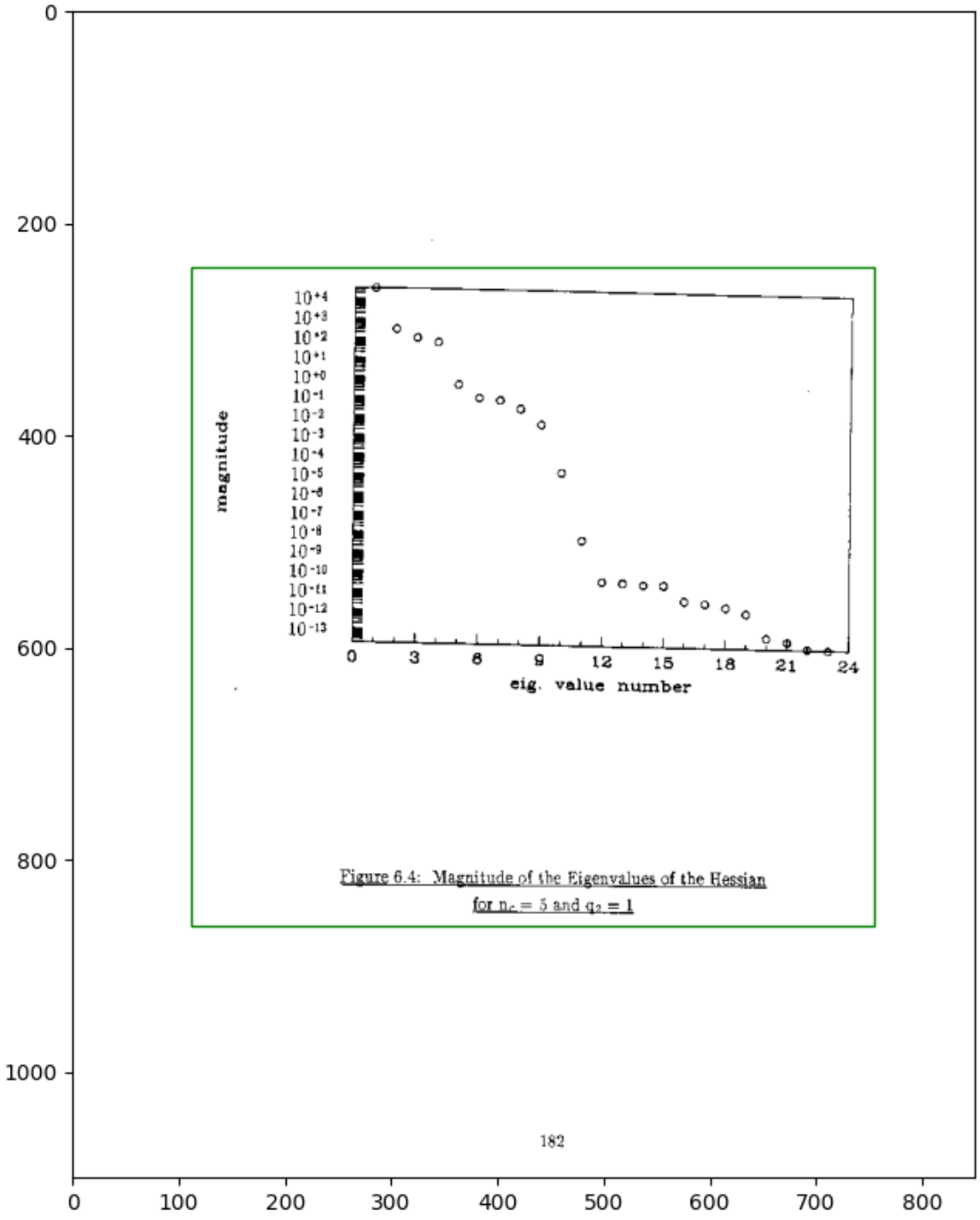


Figure 6.4: Magnitude of the Eigenvalues of the Hessian for $n_c = 5$ and $q_2 = 1$

Figure C.13: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/42427, page number: 182.

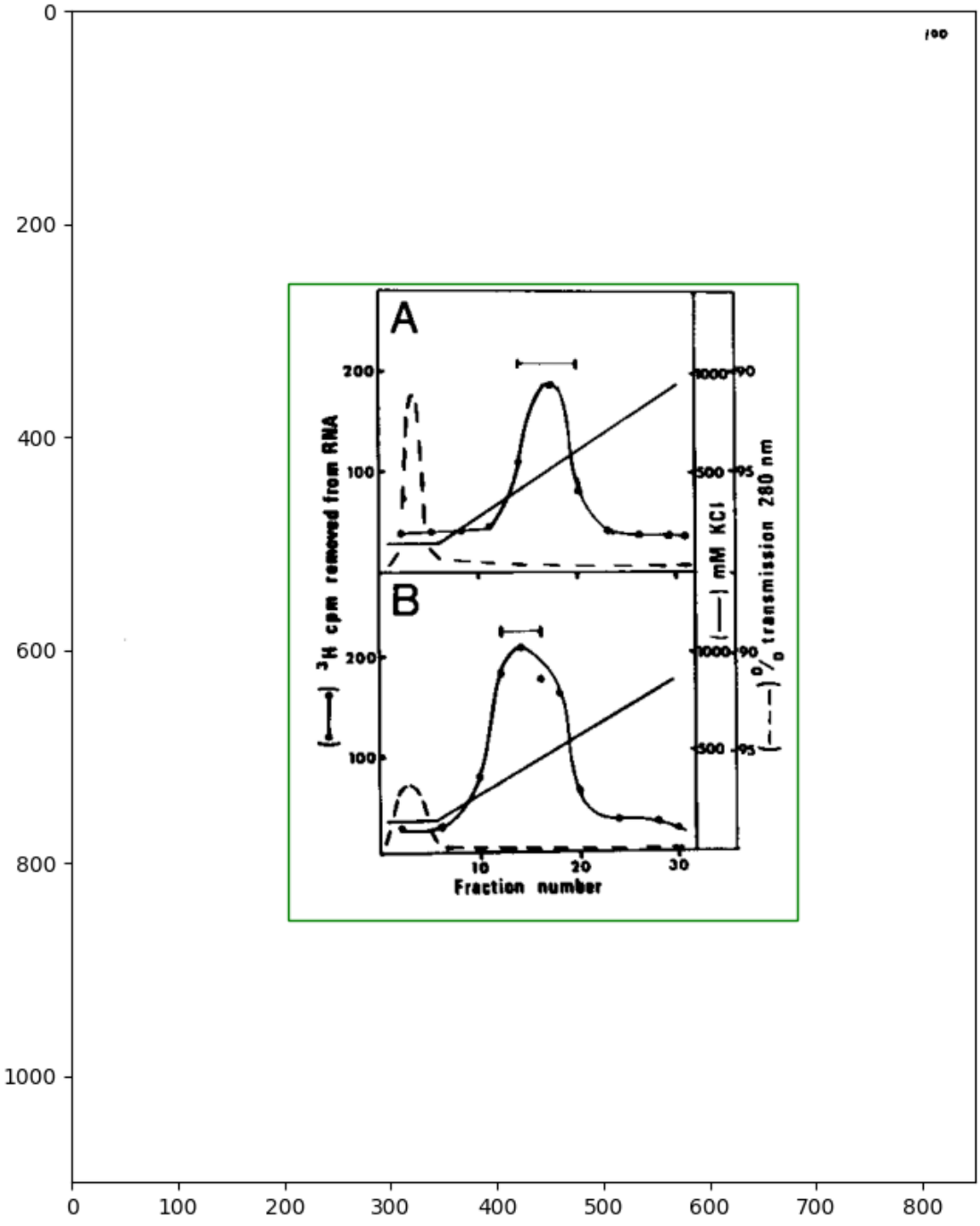


Figure C.14: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/45675, page number: 110.

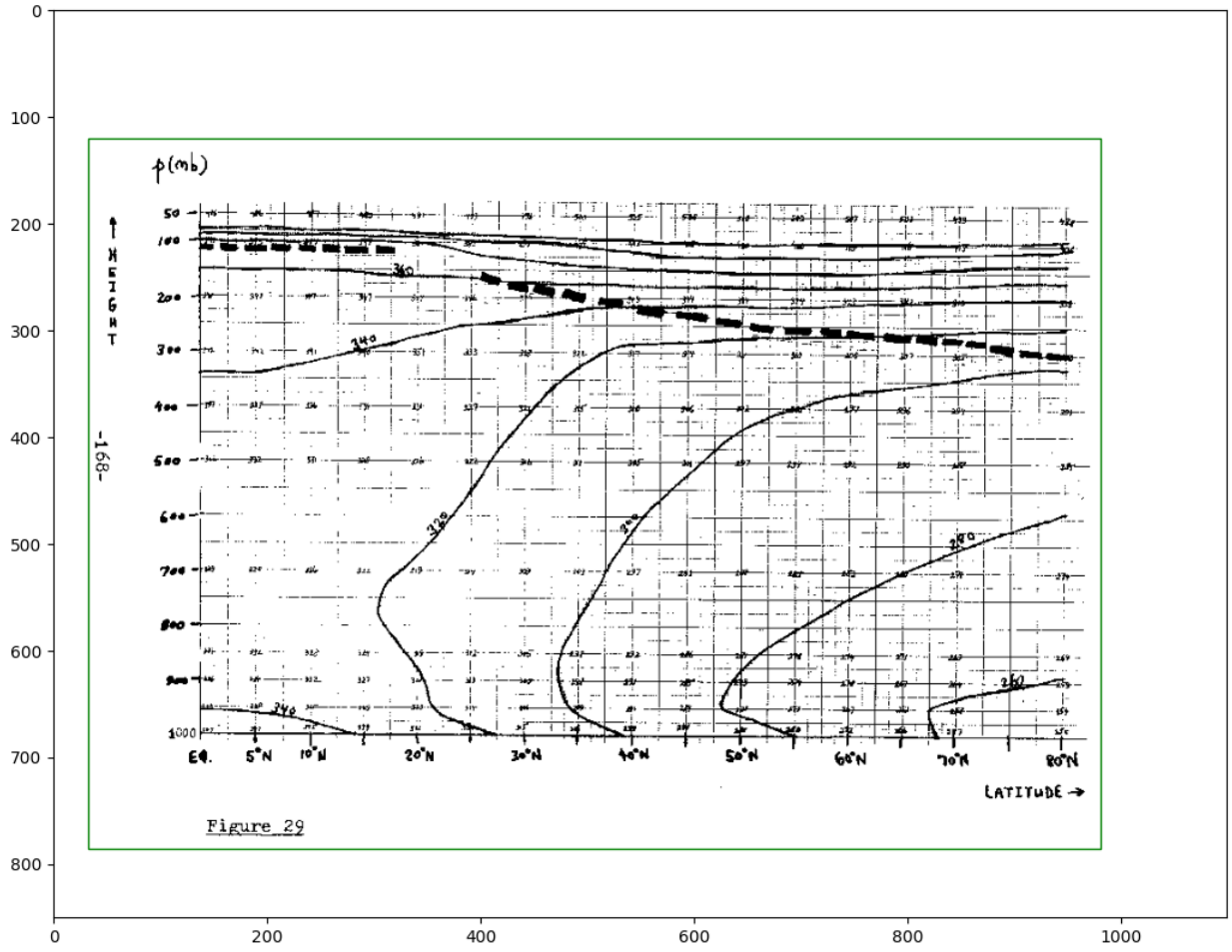


Figure C.15: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/54312, page number: 168.

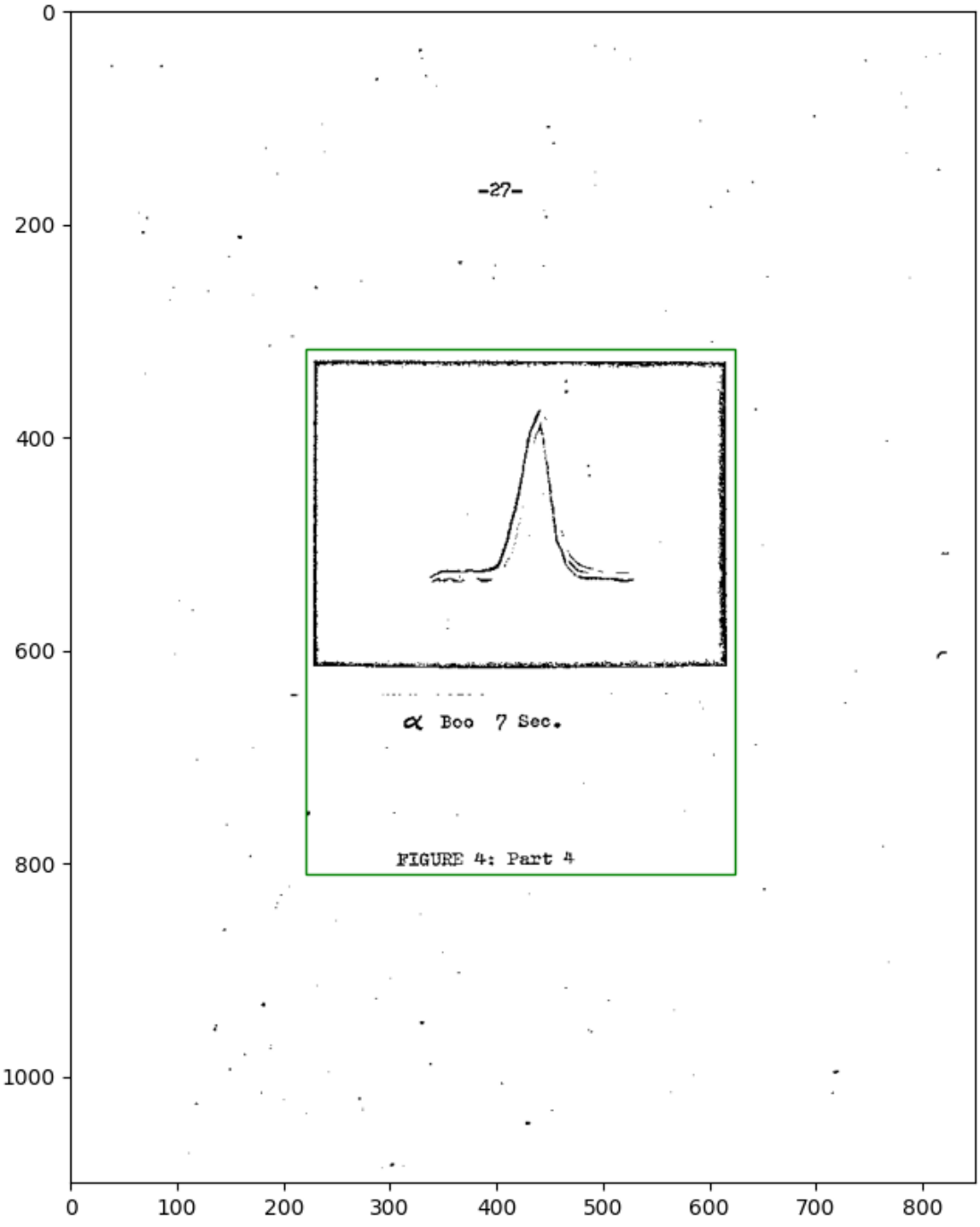


Figure C.16: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/54373, page number: 027.

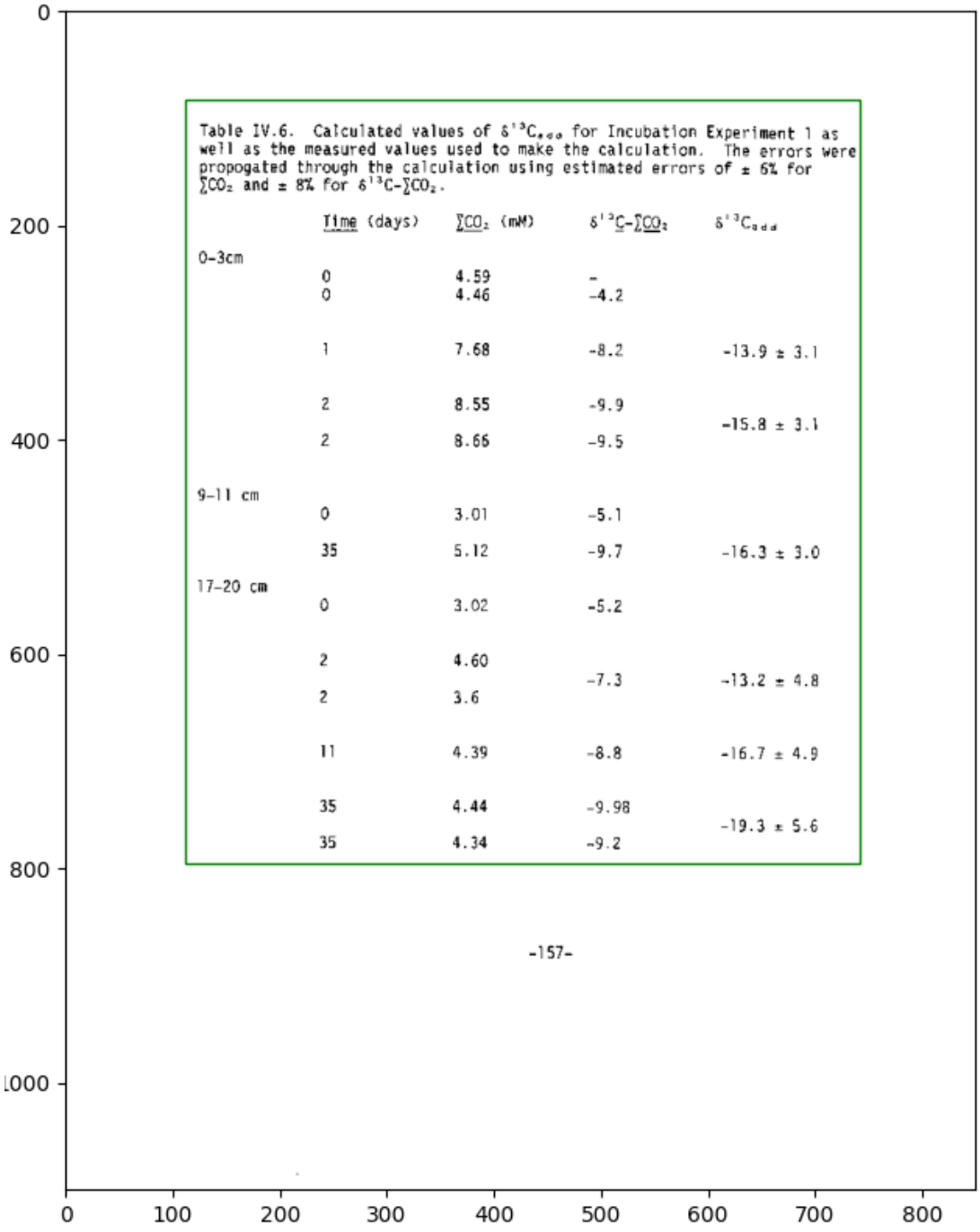


Table IV.6. Calculated values of $\delta^{13}C_{add}$ for Incubation Experiment 1 as well as the measured values used to make the calculation. The errors were propogated through the calculation using estimated errors of ± 6% for $\sum CO_2$ and ± 8% for $\delta^{13}C$-$\sum CO_2$.

	Time (days)	$\sum CO_2$ (mM)	$\delta^{13}C$-$\sum CO_2$	$\delta^{13}C_{add}$
0-3cm				
	0	4.59	-	
	0	4.46	-4.2	
	1	7.68	-8.2	-13.9 ± 3.1
	2	8.55	-9.9	-15.8 ± 3.1
	2	8.66	-9.5	
9-11 cm				
	0	3.01	-5.1	
	35	5.12	-9.7	-16.3 ± 3.0
17-20 cm				
	0	3.02	-5.2	
	2	4.60	-7.3	-13.2 ± 4.8
	2	3.6		
	11	4.39	-8.8	-16.7 ± 4.9
	35	4.44	-9.98	-19.3 ± 5.6
	35	4.34	-9.2	

-157-

Figure C.17: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/58489, page number: 157.

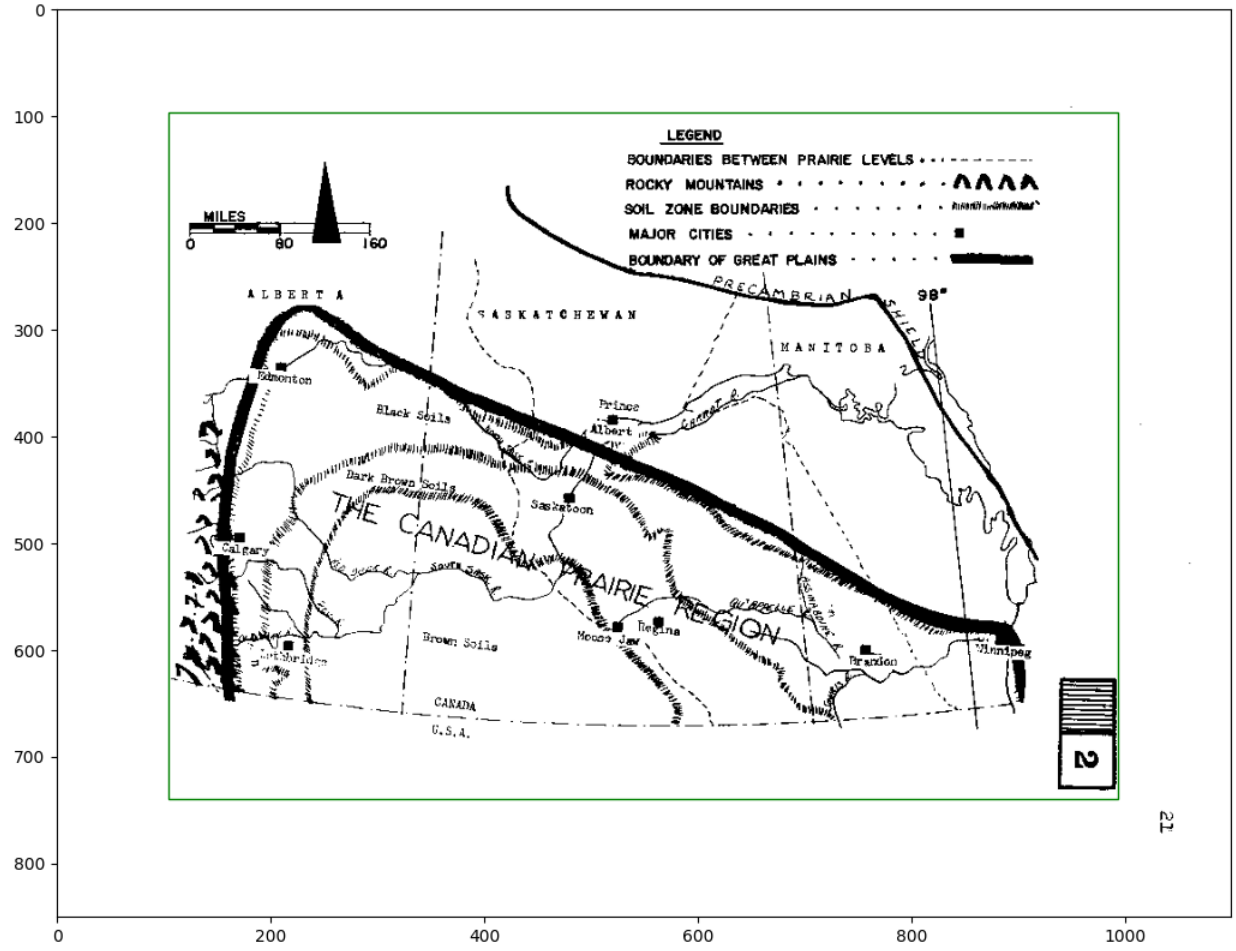


Figure C.18: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/75557, page number: 021.

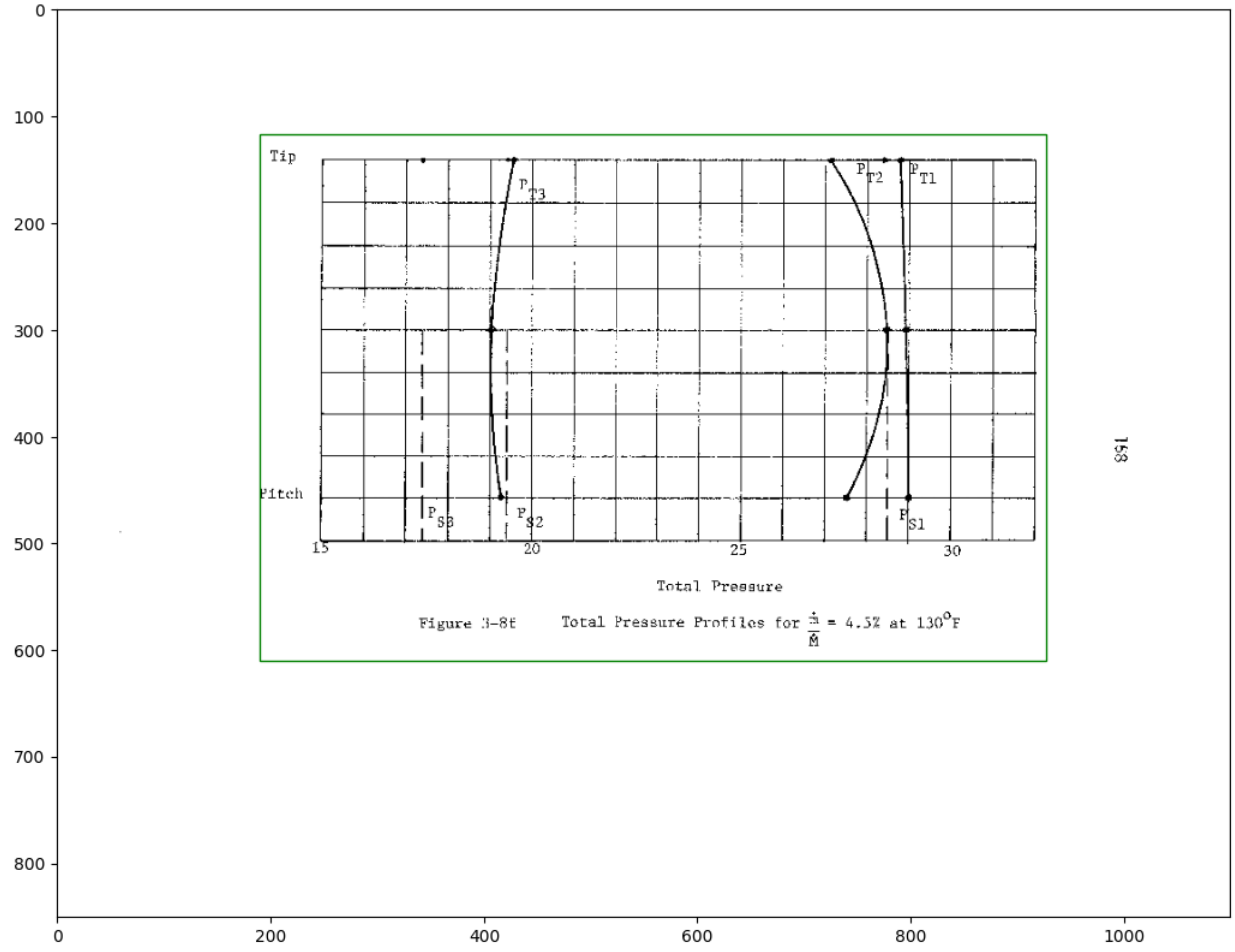


Figure 3-8f Total Pressure Profiles for $\frac{\dot{m}}{\dot{M}}$ = 4.5% at 130°F

Figure C.19: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/95535, page number: 159.

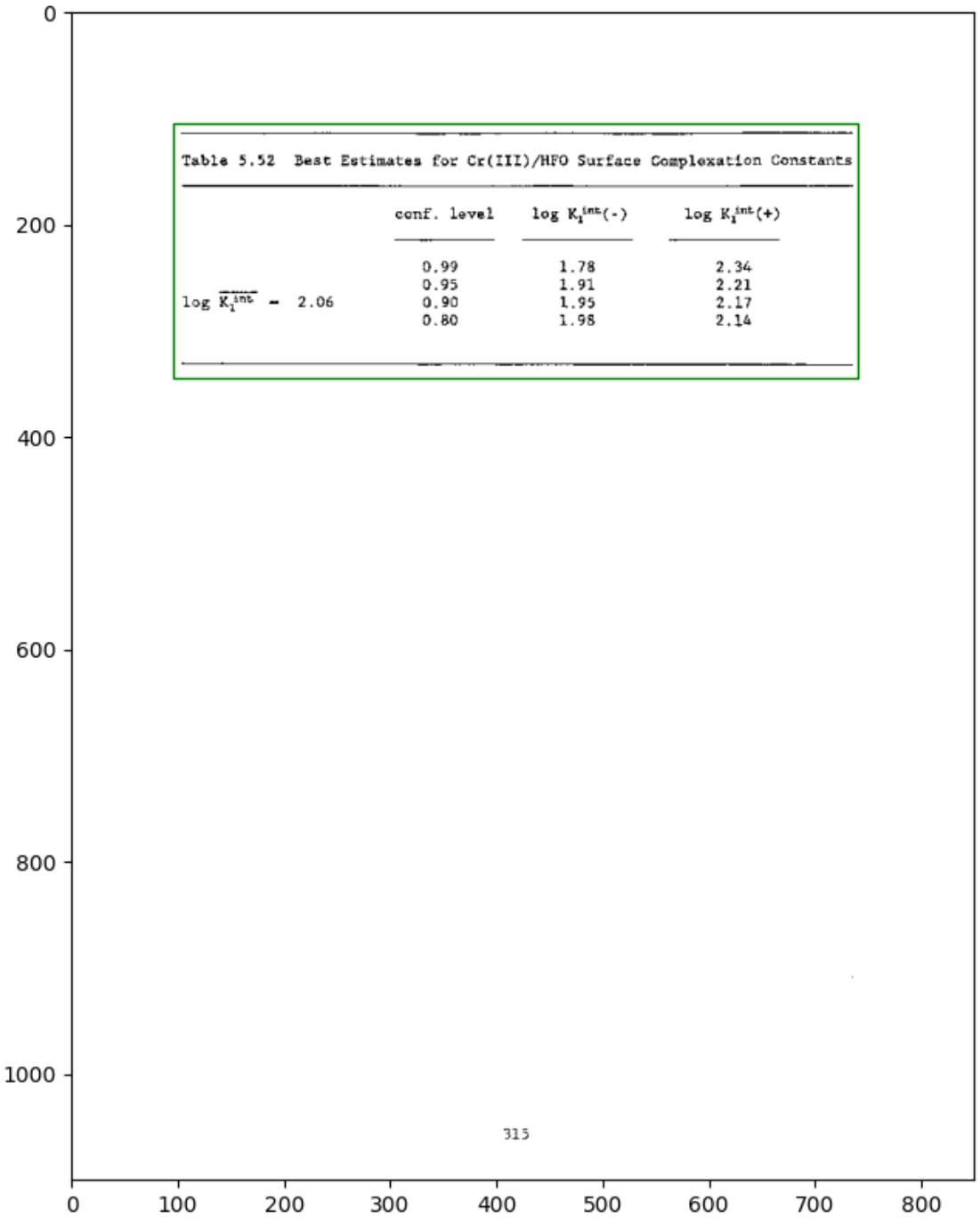


Table 5.52 Best Estimates for Cr(III)/HFO Surface Complexation Constants

	conf. level	log K_1^{int}(-)	log K_1^{int}(+)
log $\overline{K_1^{int}}$ = 2.06	0.99	1.78	2.34
	0.95	1.91	2.21
	0.90	1.95	2.17
	0.80	1.98	2.14

315

Figure C.20: Sample image with its manual annotation from the gold standard dataset. Source: https://dspace.mit.edu/handle/1721.1/97289, page number: 315.

Appendix D

Licenses

Bibliography

[1] Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Anaconda, Nov. 2016. URL https://anaconda.com. Accessed 8/14/2020.

[2] CIFAR-10 and CIFAR-100 datasets. URL https://www.cs.toronto.edu/~kriz/cifar.html. Accessed 8/17/2020.

[3] COCO - Common Objects in Context - Dataset format. URL https://cocodataset.org/#format-data. Accessed 8/17/2020.

[4] ml-lab/TensorBox. URL https://github.com/ml-lab/TensorBox. Accessed 8/14/2020.

[5] IBM100 - The Selectric Typewriter. URL http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/selectric/. Accessed 7/21/2020.

[6] LXML - Processing XML and HTML with Python. URL https://lxml.de/. Accessed 7/6/2020.

[7] DEALING WITH MEASUREMENT NOISE - Averaging Filter. URL https://web.archive.org/web/20100329135531/http://lorien.ncl.ac.uk/ming/filter/filewma.htm. Accessed 7/20/2020.

[8] The PASCAL Visual Object Classes Homepage. URL http://host.robots.ox.ac.uk/pascal/VOC. Accessed 8/17/2020.

[9] torchvision.datasets — PyTorch 1.6.0 documentation. URL https://pytorch.org/docs/stable/torchvision/datasets.html. Accessed 8/17/2020.

[10] TensorFlow Datasets. URL https://www.tensorflow.org/datasets/catalog/overview. Accessed 8/17/2020.

[11] TeX Live - TeX Users Group. URL https://www.tug.org/texlive. Accessed 8/14/2020.

[12] ultralytics/yolov5. URL https://github.com/ultralytics/yolov5. Accessed 8/14/2020.

[13] Adobe PDF Library, July 2020. URL https://dev.datalogics.com/adobe-pdf-library/. Accessed 7/1/2020.

[14] Cairo Graphics, July 2020. URL https://www.cairographics.org/. Accessed 7/3/2020.

[15] DSpace@MIT Home, June 2020. URL https://dspace.mit.edu/. Accessed 6/24/2020.

[16] pdfact, June 2020. URL https://github.com/ad-freiburg/pdfact. original-date: 2018-03-28T15:11:09Z.

[17] pdftocairo - Portable Document Format (PDF) to PNG/JPEG/TIFF/PDF/PS/EPS/SVG using cairo, July 2020. URL http://manpages.ubuntu.com/manpages/trusty/man1/pdftocairo.1.html. Accessed 7/1/2020.

[18] Poppler, July 2020. URL https://poppler.freedesktop.org/. Accessed 7/3/2020.

[19] XpdfReader, July 2020. URL https://www.xpdfreader.com/. Accessed 7/3/2020.

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal

Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[21] Md. Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Mahmudul Hasan, Brian C. Van Esesn, Abdul A. S. Awwal, and Vijayan K. Asari. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *CoRR*, abs/1803.01164, 2018. URL http://arxiv.org/abs/1803.01164.

[22] Simon Barthelmé, Hans Trukenbrod, Ralf Engbert, and Felix Wichmann. Modelling fixation locations using spatial point processes. 2012. URL http://arxiv.org/abs/1207.23701.

[23] Dr. Burrows. On Disorders of the Cerebral Circulation, and on the Connexion between Affections of the Brain and Diseases of the Heart. *The British and Foreign Medical Review*, 22(44):408–428, October 1846. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5617574/.

[24] PubMed Central. PMC open Access Subset, 2019. URL https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/. Accessed 10/9/2019.

[25] Walter Douglas Chiles. *Effect of service on automobile crankcase oils*. PhD thesis, Virginia Agricultural and Mechanical College and Polytechnic Institute, 1935. URL http://hdl.handle.net/10919/56159.

[26] Christopher Clark and Santosh Divvala. PDFFigures 2.0: Mining Figures from Research Papers. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries*, JCDL '16, pages 143–152, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4229-2. doi: 10.1145/2910896.2910904.

[27] A. Dutta, A. Gupta, and A. Zissermann. VGG image annotator (VIA). http://www.robots.ox.ac.uk/ vgg/software/via/, 2016. Version: 2.0.9.

[28] Abhishek Dutta and Andrew Zisserman. The VIA Annotation Software for Images, Audio and Video. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6889-6/19/10. doi: 10.1145/3343031.3350535. URL https://doi.org/10.1145/3343031.3350535.

[29] Alfred M. Gossage. The Automatic Rhythm of the Heart. *British Medical Journal*, 2(2452):1818–1821, December 1907. ISSN 0007-1447. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2358859/.

[30] hackerb9. How to extract vectors from a PDF file?, July 2020. URL https://superuser.com/a/884445/978042. Accessed 8/17/2020.

[31] Matthias Hansen, André Pomp, Kemal Erki, and Tobias Meisen. Data-Driven Recognition and Extraction of PDF Document Elements. *Technologies*, 7(3):65, 2019.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[34] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, Weng Chi-Hung, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug GitHub repository, 2019. URL https://github.com/aleju/imgaug. Accessed 9/25/2019.

[35] Pavel K. PDFEdit, June 2020. URL https://github.com/nullishzero/PDFEdit. original-date: 2012-03-09T11:35:33Z.

[36] Sampanna Kahu. SampannaKahu/deepfigures-open. URL https://github.com/SampannaKahu/deepfigures-open. Accessed 8/14/2020.

[37] Renu Khandelwal. Evaluating performance of an object detection model, January 2020. URL https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b. Accessed 6/24/2020.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[39] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi: 10.1002/nav.3800020109. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[40] Benjamin Charles Germain Lee, Jaime Mears, Eileen Jakeway, Meghan M. Ferriter, Chris Adams, Nathan Yarasavage, Deborah Thomas, Kate Zwaard, and Daniel S. Weld. The Newspaper Navigator Dataset: Extracting And Analyzing Visual Content from 16 Million Historic Newspaper Pages in Chronicling America. *ArXiv*, abs/2005.01583, 2020. URL https://arxiv.org/abs/2005.01583.

[41] Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, Ming Zhou, and Zhoujun Li. TableBank: Table Benchmark for Image-based Table Detection and Recognition. *CoRR*, abs/1903.01949, 2019. URL http://arxiv.org/abs/1903.01949.

[42] Patrice Lopez. GROBID: Combining Automatic Bibliographic Data Recognition and Term Extraction for Scholarship Publications. In Maristella Agosti, José Borbinha, Sarantos Kapidakis, Christos Papatheodorou, and Giannis Tsakonas, editors, *Research and Advanced Technology for Digital Libraries*, pages 473–474, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04346-8. doi: 10.1007/978-3-642-04346-8_62.

[43] Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.*, 2014(239), March 2014. ISSN 1075-3583.

[44] George Ralph Mines. On dynamic equilibrium in the heart. *The Journal of Physiology*, 46(4-5):349–383, July 1913. ISSN 0022-3751. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1420430/.

[45] Heiko Oberdiek. fonts - \textbf does not work with cmvtt style. URL https://tex.stackexchange.com/a/415936/218782. Accessed 8/17/2020.

[46] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134. Accessed 7/21/2020.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and

R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. Accessed 8/14/2020.

[48] Luis Perez and Jason Wang. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *CoRR*, abs/1712.04621, 2017. URL http://arxiv.org/abs/1712.04621.

[49] Lorien Y. Pratt. Discriminability-based transfer between neural networks. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, page 204–211, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. ISBN 1558602747.

[50] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[51] Zhibin Ren, Xingyuan He, Haifeng Zheng, and Hongxu Wei. Spatio-Temporal Patterns of Urban Forest Basal Area under China's Rapid Urban Expansion and Greening: Implications for Urban Green Infrastructure Management. *Forests*, 9(5):272, May 2018. ISSN 1999-4907. doi: 10.3390/f9050272. URL http://dx.doi.org/10.3390/f9050272.

[52] Roy and J. G. Adami. Remarks on Failure of the Heart from Overstrain. *British Medical Journal*, 2(1459):1321–1326, December 1888. ISSN 0007-1447. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2198364/.

[53] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *CoRR*, abs/1312.6229, 2013. URL http://arxiv.org/abs/1312.6229.

[54] Noah Siegel, Nicholas Lourie, Russell Power, and Waleed Ammar. Extracting Scientific Figures with Distantly Supervised Neural Networks. *CoRR*, abs/1804.02445, 2018. URL http://arxiv.org/abs/1804.02445.

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[56] Martin A Tanner. *Tools for statistical inference: observed data and data augmentation methods*, volume 67. Springer Science & Business Media, 2012. doi: 10.1007/978-1-4684-0510-1.

[57] Sebastian Thrun and Lorien Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, USA, 1998. ISBN 0792380479.

[58] Cornell University. arXiv Bulk Data Access - Amazon S3 | arXiv e-print repository, 2019. URL https://arxiv.org/help/bulk_data_s3#src. Accessed 10/3/2019.

[59] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *CoRR*, abs/1703.10593, 2017. URL http://arxiv.org/abs/1703.10593.