Object Detection

Final Report

Kecheng Zhu

Zachary Gager

Shelby Neal

Jiangyue Li

You Peng

CS 4624: Multimedia, Hypertext, and Information Access

Virginia Tech

Blacksburg, VA 24061

May 9, 2022

Instructor: Dr. Edward A. Fox

Client: Aman Ahuja

Table of Contents

1 Table of Figures

2 Table of Tables

3 Executive Summary / Abstract

Electronic theses and dissertations (ETDs) contain valuable knowledge that can be useful in a wide range of research areas. To effectively utilize the knowledge contained in ETDs, the data first needs to be parsed and stored in an XML document. However, since most of the ETDs available on the web are presented in PDF, parsing them is a challenge to make their data useful for any downstream task, including question-answering, figure search, table search, and summarizing.

For information search and extraction, contextual information is needed to perform these tasks. However, such semantic information is hidden in PDF documents. In contrast, XML can explicitly share semantic information. The structure within XML documents can enforce semantic continuity within the tag elements. Accordingly, knowledge graphs can be more easily built from XML, rather than PDF, representations.

The goal of this project was to extract different elements of scholarly documents such as metadata (title, authors, year), chapter headings and subheadings, equations, figures (and captions), tables (and captions), and paragraphs, and then package them into an XML document. Subsequently, a pipeline responsible for the conversion and a dataset to support the object detection step was developed.

Over the semester, 200 ETDs, both born-digital and scanned, were annotated using a online tool called RoboFlow. A model based on Facebook's open-sourced object detection model, Detectron2, was trained with the created dataset. Besides that, a pipeline that utilizes the model has been built that converts an ETD in PDF into an XML document, which can then be used for future downstream tasks and HTML for visualization. A dataset consisting of 200 annotated ETDs and a working pipeline were delivered to the client. From the project, the Object Detection Team learned numbers of libraries related to the task, built a sense of the importance of version control, and understood how to split a large task into smaller and more approachable pieces.

4 Introduction

Electronic theses and dissertations (ETDs) are scientific research works with a suitable degree of originality, and are a focus of collection and utilization. They represent different levels of learning and are among the most important sources of scholarship and information. The general structure of a dissertation is relatively fixed; it contains some main items, where each main item has a certain purpose, is written with certain language characteristics, and satisfies specific requirements.

A born-digital ETD is an ETD with text embedded in its PDF file. Accordingly, the text elements can be extracted from the original ETD. This means that if the data of digital ETDs can be properly parsed and stored in an XML document, then this valuable knowledge can be put to use in a variety of fields.

Unfortunately, however, there is no standard format for most of the ETDs that are found nowadays, and ETDs published by different universities and organizations often have their own formats. For example, certain institutions may prefer to label their first section as "introduction", while others prefer "background information". In addition, the quality of the documents varies, as ETDs can either be scanned or born-digital. Consequently, in scanned ETDs, some of the pages may not be aligned and the text in the page may be crooked, or text on the edges could be cut off.

Therefore, it is a challenge for researchers to correctly parse long PDF documents. In terms of providing access to valuable knowledge for effective use of ETDs, work needs to done to add value to ETDs by enabling question-answering, figure search, table search, and summarization. The main purpose of this project is to develop an algorithm that can identify the different parts of ETDs and help with these activities.

For this object detection project, a model needs to be built that is trained on a large dataset of ETDs. The selected model is a neural network that is designed for object detection tasks. There are two primary methods when training a neural network model for a new task. The first approach is to train the model from scratch and randomly assign weights to all parameters before training. The second approach is often called transfer learning. In transfer learning, the neural network starts with weights trained on a similar problem [13]. This can decrease the required size of the training dataset and the time it takes to train for the new task. After the training is done, the model can also parse other files similar to the training ETDs, in a reasonable way.

Therefore, the dataset is a very important part of the project. The accuracy of the whole project depends largely on the accuracy of the dataset. A new dataset (ETD-OD) ETD-Object Detection is introduced, and it is a dataset for object detection in long scholarly documents like ETDs. Images from 200 ETDs are manually annotated by humans with bounding boxes around each of the elements. The entire annotation process is handled by a three-person labeling group. To ensure the accuracy of the annotation, a cross-processing approach is used. The approach involves separating the labeling and dataset import steps, with different team members taking care of different parts of the process. After labeling a part of the data, someone else reviews it and adds the labelled data into the dataset. While the dataset is being created, a model is also developed to accept an ETD's PDF as input and then convert it to an XML representation.

The aim of this project is to develop a pipeline that accepts an ETD's PDF as input; converts its pages into images; uses the trained object detection model to split it into labelled page elements; converts the page elements into either text (through OCR) or images; and, lastly, uses the parsed text and images to save the processed data using XML.

5 Requirements

5.1 Object Detection Pipeline

To convert a PDF file to a format where document elements/objects can be identified, a pipeline is needed. First, it is necessary to convert PDF pages to images as an intermediate format so that the visually based Machine Learning (ML) model can classify different elements and extract them. Among these classified objects, elements such as chapter title and paragraph go through an optical character recognition (OCR) step and are written into an XML file. Elements such as figures and tables are stored as images in a file system, and a reference to their location in the file system is saved in an output XML file.

5.2 Dataset Development

A set of ETDs, including both born-digital and scanned PDFs, is provided by the client. For the dataset to be developed, the pages of all the PDFs need to first be converted to images. As such, four batches, each including 50 ETDs, were randomly selected from the ETDs provided, and converted into a set of images. Then this dataset, containing more than 20000 images, was labeled with the following tags[1]:

- Abstract: Abstract Heading, Abstract Text
- Metadata: Title, Author, Date, University, Committee, Degree
- Chapter: Chapter Title, Section, Paragraph, Figure, Table, Equation, Algorithm, Definition, Footnote
- Table of Contents: List of Contents, TOC Title
- Reference: Reference Text, Reference Heading

[1]All tags were provided by the client, along with definitions of what each entails. The specific definition of each tag is provided in Table 2.

6 Methodology

This section shows a breakdown of the requirements into smaller and more approachable problems. This section is a modification from the methodology assignment.

6.1 Goal

Object Detection - The ability to use an ML model to detect different elements in an Electronic Thesis or Dissertation (ETD). Convert ETDs into XML - The ability to extract sections of each page based on detected results, convert those sections to text or reference, and generate an XML document.

6.2 Sub-tasks

1. Object Detection
   (a) Create data set to train the detection model
      i. Convert PDFs into images
      ii. Label converted images
      iii. Analyze dataset distribution
   (b) Construct and train the ML model
      i. Model selection
      ii. Train the selected model
         A. Preliminary training
         B. Train model on provided server
      iii. Evaluate the model's results
2. Transfer ETDs into XML
   (a) Extract elements from ETDs with the ML model
   (b) Convert extracted text elements to text
      i. Convert extracted text elements from a born-digital ETD to text
      ii. Convert extracted text elements from a scanned ETD to text
   (c) Save extracted image elements to a file system and add their paths to the XML file
   (d) Write text and image paths into the XML file

6.3 Implementation Based Services

Service ID	Service Name	Input file name(s)	Output file name	Libraries; Functions; Environments
1	Convert PDFs to images	Any folder	<PDF name>_<page number>.png	pdf2image
2	Label converted images	*.png	*.zip/URL	RoboFlow
3	Analysis dataset distribution	N/A	N/A	RoboFlow
4	Model selection	N/A	*.pth	Detectron2
5	Preliminary training	*.zip/URL, *.pth	model_final.pth	Google Colab
6	Training on larger dataset	*.zip/URL, *.pth	model_final.pth	CUDA-enabled server
7	Evaluate model results	model_final.pth	result.csv	COCO evaluator
8	Extract elements from page images	<filename>_<page number>.png, model_final.pth	Temporary data structure	Detectron2, CV2
9	Extracted text elements to text	Temporary data structure	Temporary text	PyMuPDF, Pytesseract
10	Save extracted images	Temporary data structure	<img_path>.png	CV2
11	Create XML file	Temporary text, <img_path>.png	*.xml	XML Module

Table 1: Implementation Service Table

6.4 Pipeline Diagram and Workflow

The pipeline diagram in Figure 1 provides an outline of the main services from Section 6.3. The pipeline takes in one PDF as input, and Service 1 converts the PDF's pages into images. The results are sent to a trained object detection model. In Service 8, the model outputs a bounding box for each element in each page image. These are then used to crop the elements from their original image. If the cropped element is labelled as a text element (like title or paragraph), the image needs to be converted to text by Service 9. Otherwise, the cropped element is saved as an image by Service 10. Lastly, Service 11 creates an XML file with the text elements from Service 9 and the paths to the cropped images from Service 10.

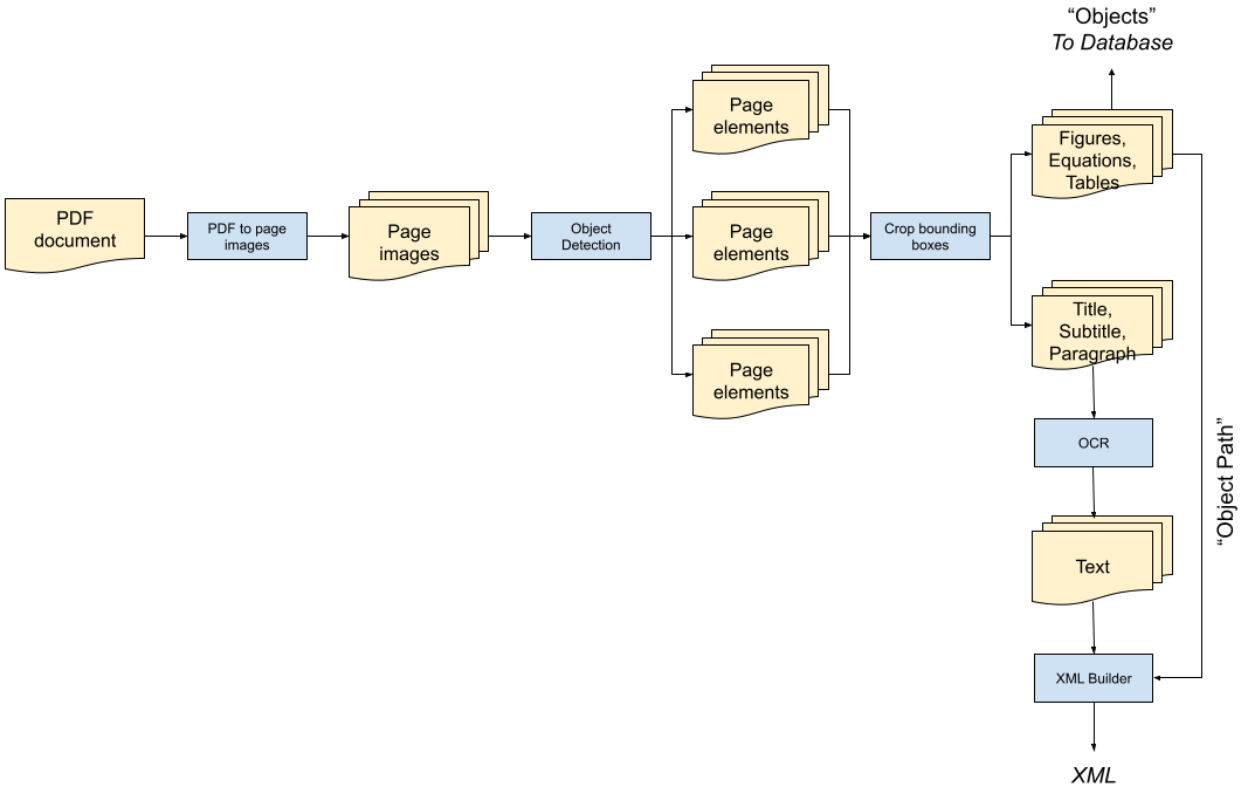

Figure 1: Pipeline Diagram

7 Implementation

7.1 PDF to Image

The first step was to convert PDFs in a folder to a set of image files. This was accomplished with the PDF to image script `pdf2img_converter.py` which needs a directory path as input. The converted images were stored in a set of folders with names corresponding to their original PDF names, and each image was named as <pdf name>_<page number>.png.

This script makes use of `convert_from_path` function in the `pdf2img` package and the `os` Python module is used to convert a set of PDFs in a directory to a set of images in a folder. This script also makes sure that the converted images are not resized or zoomed, as the client requested.

Figure 2: PDF to Image Input / Output

7.2 Dataset Creation

7.2.1 List of Labelled Elements in ETDs

In order to cover all kinds of ETDs, both Scanned PDFs (created through a typewriter, printer, or hand written and then scanned electronically) and born-digital PDFs (digitally created PDFs with embedded text), were collected and labeled. Collected ETDs were randomly separated into smaller batches, with each batch containing 50 ETDs. The ETDs were then labeled with the following categories and stored in COCO data format, as discussed in Section 7.2.3.

1. All types of Table of Contents (Table of Contents, Table of Figures, List of Figures, etc.) were regarded as Table of Contents. They can be differentiated based on the TOC title.
2. Appendix was regarded as a chapter. It can be identified based on its corresponding chapter title.
3. For documents with only one chapter, regard different sections like introduction, experiments, conclusion, etc. as a chapter.
4. For documents with multiple chapters, annotate headings of different sections / subsections clearly with the “Section” label.

No.	Tag name	Description
1	algorithm	algorithms in the field of computer science and math
2	chapter title	title of each chapter
3	chapter subheading	includes all subheadings that is not a chapter title
4	degree	degree of the author
5	list of content heading	list of content section title
6	list of content text	list of content text
7	title	title of the ETD
8	author	author of the ETD
9	committee	the committee approved the ETD
10	date	month and year the paper is published
11	equation	equations
12	equation number	label for equations
13	figure	includes figures and hand drawn chart
14	figure caption	includes caption for figures and hand drawn chart
15	foot note	footnote in ETDs
16	page number	page number in ETDs
17	reference heading	reference section title, e.g., bibliography
18	reference text	reference in ETDs
19	paragraph	paragraphs in ETDs
20	table	includes table, list of figures, list of tables
21	table caption	includes caption for table, list of figures, list of tables
22	university	author's university

Table 2: List of Tags Labelled

7.2.2 Train / Test / Validation Split

After an annotated image dataset was created (for example, a dataset containing annotated images for scanned PDFs), the resulting images were split into training / validation / testing datasets with 70% / 10% / 20% of the whole dataset, respectively.

- *Training Dataset:* All of the training dataset images and their corresponding annotations were used for training the Detectron2 model, as discussed in Section 7.3.
- *Testing Dataset:* All of the testing dataset images and their corresponding annotations were used for calculating evaluation metrics of the Detectron2 model, as discussed in Section 8.1.
- *Validation Dataset:* The validation dataset was purely used for visually validating the model results. For example, an image would be given to the model, the model labels the image, and then the visualization of the model predictions would be compared to the true annotations of the original image.

The images were split with no acknowledgement as to what ETD the specific images came from. For example, for a single ETD, the first page's image can go to the training dataset and the second page's image can go to the testing dataset.

7.2.3 COCO Data Format

After the train / test / validation split, all three datasets were formatted according to the COCO data format. A single COCO formatted dataset [6] is stored in a flat file directory with all the `.png` image files and a single `_annotations.coco.json` file containing the annotations for all the images.

The COCO data format is used to describe each object, like paragraphs and chapter titles, across multiple images in a dataset. Each object contains a series of fields, as shown in Figure 3. Every object has a unique ID, an ID for the image the object is located in, and an ID for the category the object represents. Additionally, each object is defined by the coordinates of a bounding box. The segmentation field is also often used to mask an object that has a well-defined parameter, like an object that represents a person. However, in the ETD dataset, objects like paragraphs and chapter titles do not have a well-defined parameter that could define the object better than a bounding box. As such, segmentation was not used in the ETD dataset. The `iscrowd` parameter is also often used when an object represents multiple instances of the same object, like an object representing multiple people. However, this does not happen with any of the objects in the ETD dataset, so this parameter is always set to 0.

```
annotation{
   "id"                : int,
   "image_id"          : int,
   "category_id"       : int,
   "segmentation"      : RLE or [polygon],
   "area"              : float,
   "bbox"              : [x,y,width,height],
   "iscrowd"           : 0 or 1,
}

categories[{
   "id"                : int,
   "name"              : str,
   "supercategory"     : str,
}]
```

Figure 3: Parameters in COCO Data Format for Object Detection Tasks

7.3 Object Detection Model

The DocBank repository consists of multiple neural networks that were trained on document images [4]. One such neural network, X101, was trained with the Detectron2 Base-RCNN-FPN model. Before training on the custom dataset, the pretrained weights from DocBank's X101 were loaded into Detectron2's Base-RCNN-FPN model. This method of loading in weights from a similar problem is often called transfer learning,

and it can decrease the amount of data the model needs when learning the new task. After loading in the pretrained weights, the model was trained with the training datasets described in Section 7.2. Next, the resulting weights of the model were stored in a PyTorch model file with the `.pth` file extension, and the trained model was used in the PDF to XML pipeline from Section 6.4. In the pipeline, the model's input was an image of a page from an ETD, and its output was a list of labelled elements from the image.

7.3.1 Repeat Factor Training Sampler

ETDs can have hundreds of paragraphs but only one title, which leads to a serious imbalance in the data categories. For paragraphs, pictures, and tables, there were thousands of samples in the data. But for title, author, and abstract, there were less than 300 samples. Because of this, a Repeat Factor Training Sampler was also used to conduct the sampling during training. The Repeat Factor Training Sampler increases the sampling rate for tail class instances by oversampling images containing these categories. After a `cfg` item is created for the Detectron2 model, the Repeat Factor Training Sampler can be specified with

```
cfg.DATALOADER.SAMPLER_TRAIN = "RepeatFactorTrainingSampler"
cfg.DATALOADER.REPEAT_THRESHOLD = 0.001
```

The `REPEAT_THRESHOLD` controls the point at which oversampling kicks in. For example, when `REPEAT_THRESHOLD = 0.001`, categories that appear in less than 0.1% of images are oversampled. It is possible to test varying percentages for this parameter. Researchers at Facebook AI Research discovered that, for their Large Vocabulary Instance Segmentation dataset, a threshold of 0.001 produced the best evaluation results [3]. Before updating this parameter, the Detectron2 default is to sample each image with equal likelihood. Section 8 discusses the evaluation results before and after updating this parameter.

7.3.2 Model Input and Output

After the model was trained, the model is then used in the pipeline, as discussed in Section 6.4. In the pipeline, the input to the model is a single page from an ETD. The output of the model is a list of bounding boxes and a list of predicted labels. The bounding boxes are formatted in `(x, y, width, height)` format, corresponding to the input image's dimensions. Each bounding box has a corresponding predicted label, indicating what type of element the bounding box represents (paragraph, figure, caption, etc.), as discussed in Section 7.2.1.

It is important to note that the bounding boxes are, by default, not sorted in any particular order by the model. An ETD is typically written in one column, so its sections are ordered by their y-coordinate. As such, after the model makes its predictions in the pipeline, the bounding boxes are sorted by their top left y-coordinate.

7.4 Parsing the ETD

After a page of the ETD is run through the model, the parsing step used the model's output to parse the page into both text and image elements. The input to the parsing step is an image of a page and the model's

predictions of that page. The output of the parsing step is a list of raw text and image elements with their corresponding section labels (paragraph, figure, etc.) for the entire ETD.

7.4.1 Parsing Images

Bounding boxes labeled as an algorithm, equation, figure, or table are stored as images. Because the bounding box represents the coordinates of the element, the bounding box is used to crop the image from the image of the original page. Next, the cropped image is saved using the `cv2` package in Python.

7.4.2 Parsing Text from a Digital ETD

Born-digital ETDs are ETDs that have embedded text in their PDF. If the current ETD being parsed was originally a digital ETD, the text elements can be extracted from the original ETD's PDF. This can be performed using the Python package `PyMuPDF`, which allows a user to extract embedded text from a PDF file using a bounding box. It is important to note that the coordinates of the model's bounding box are in terms of the page's converted image. So, before `PyMuPDF` can be used, the coordinates need to be converted into PDF coordinates.

7.4.3 Parsing Text from a Scanned ETD

Scanned ETDs are ETDs that have minimal embedded text. If an ETD does have any embedded text, the embedded text usually indicates metadata of the ETD, and cannot help with extracting text like paragraphs and chapter titles. As such, the text elements from the ETD have to be converted to raw text through Optical Character Recognition (OCR). First, the bounding box is used to crop the image of the text element from the page. Next, that cropped image is run through OCR using the Python package `Pytesseract` to obtain the raw text.

7.5 XML Creation

The parsing step creates a Python list, where each element is a tuple with format `(element_label, element_text_or_image, page_number)`. These tuples are sorted by the order they appear in the ETD, from top to bottom. A number of rules are used for converting this data structure into an XML file - for example, elements have to be embedded under their corresponding chapters, and captions have to be embedded with their corresponding figures; see Code Listing 1.

Code Listing 1: XML ETD Representation

```
<etd>
    <front>
        <title> ETD Title </title>
        ...
    </front>
    <body>
        <chapter>
            ...
            <chapter_subheading>
            ...
            </chapter_subheading>
            <paragraphs>
            ...
            <paragraphs>
            ...
        </chapter>
        ...
    </body>
    <back>
        <reference_heading></reference_heading>
        <reference_text></reference_text>
    </back>
</etd>
```

The `<front>` tag stores metadata for the ETD with the corresponding labels. If a certain metadata tag is not found in the parsed data structure, the tag is still added, but with no text. For example, if a title is not found, then `<title />` is still added. The `<body>` tag stores chapters and chapter subheadings. Each chapter and chapter subheading has six elements associated with it - paragraphs, figures, tables, algorithms, equations, and unused captions. Each one of these six elements will have 0 or more subelements. For example, a `<paragraphs>` tag may have zero or more `<p>` subelements. Lastly, the `<back>` tag stores the references of the ETD.

7.5.1 Associating Elements

As shown in the XML output format in Listing 1, many elements are associated as children of other elements. For the `<front>` and `<back>` tags, their associated children are searched for through the parsed output data structure one-by-one. If multiple instances of a certain metadata tag are found, for example if multiple titles are found, the first one that comes chronologically is used. However, the chapters and chapter subheadings, along with their individual elements like paragraphs and figures, have to be associated with each other based on how they appear in the parsed data structure. As such, the following rules are applied when linearly iterating through the parsed data structure:

1. Some ETDs have the same chapter title as a header on multiple pages. A new chapter title is added only if its text does not match the previous chapter title.
2. A chapter subheading is associated with the last found chapter title.
3. If a chapter subheading is found before a chapter title, then it is assumed that the model did not detect the chapter title. An empty `<chapter>` tag is created for the chapter subheading.

4. Every time a `<chapter>` or `<chapter_subheading>` is created, it is given the following six empty children: `<paragraphs>`, `<figures>`, `<tables>`, `<equations>`, `<algorithms>`, and `<unused_captions>`.

5. Paragraphs, figures, tables, equations, and algorithms are added to the last created chapter or chapter subheading. If a chapter has not been created yet, then an empty `<chapter>` tag is created for the element.

6. Captions, such as figure captions, table captions, and equation numbers, are associated with a figure if a figure came directly before or after it in the parsed data structure. However, the model may mislabel a caption or not detect a figure. As such, if there is no figure directly before or after the caption, the caption is added to `<unused_captions>`.

7.6 XML Visualization

XML is hard to read for any human related work, so the output XML is converted to HTML through a Python script. Texts are resized based on tags. The $< h1 >$ tags is for Titles; the $< h2 >$ tag is for Chapter titles, abstract heading, reference heading, Table of Contents heading; the $< h3 >$ tag is for chapter subheadings. For each chapter, all images, tables, algorithms, etc. (those stored in image format) were re-organized at the end of each paragraph, with their designated captions below them. A sample converted XML is shown in Figure 4.

Association of Working Alliance and Parenting Stress for Mothers of Toddlers At-Risk for Autism Spectrum Disorder

Association of Working Alliance and Parenting Stress for Mothers of Toddlers At-Risk for Autism Spectrum Disorder

Vattuone, Cristiana

UCLA

for the degree Master of Arts in Education

committee Not found

2013

ABSTRACT Ï¿½ OF Ï¿½ THESIS Ï¿½ Association of Working Alliance and Parenting Stress for Mothers of Toddlers At-Risk for Autism Spectrum Disorder

Introduction: Parents of children with autism spectrum disorder (ASD) consistently report elevated levels of parenting stress. The complexities associated with raising a child with ASD put parents at greater risk, highlighting the importance of understanding potential stressors and protective factors that impact parental wellbeing. As the prevalence of ASD continues to increase, children are being screened and identified at earlier ages. Still, little is understood about parents of children at-risk and the factors associated with parenting stress within this population. The purpose of this study was to examine stress profiles for parents of very young children at-risk, and to examine the working relationship or alliance between parents and early interventionists providing a research based intervention program.

TABLE OF CONTENTS

Chapter title not detected

Methods: This study utilized a working alliance framework to examine the association of alliance on lowering parental stress levels over the course of a 12-week parent mediated early intervention project. 66 toddlers at risk for autism were randomized into 12 sessions of a parent-mediated intervention group or 4 sessions of a parent education group. 45 of the 66 participants were included in the current sample. Parenting stress was measured at two time points pre and post treatment, and working alliance was measured post treatment. Results: Findings suggest that the caregivers in this study who participated in a parent-mediated intervention for their toddler at-risk reported on average clinical levels of parenting stress, as has been reported by parents of older children with ASD. Findings also showed that caregivers who participated in a parent-mediated treatment condition demonstrated higher alliance than the monitoring group. Alliance was marginally associated with lower parenting stress at the end of treatment. Conclusion: Future studies should examine alliance and parenting stress in larger samples as alliance could be an important factor in lowering parenting stress for families of at risk toddlers who are engaged in early intervention.

Introduction Parenting a child with a developmental disability can present its challenges, and families of children with autism spectrum disorder (ASD) may be particularly vulnerable. Parents of children with ASD consistently report higher levels of stress compared to parents of children with other developmental disabilities, such as Down syndrome and fragile x syndrome. These parents are also more likely to experience psychological symptoms of depression and anxiety (Baker-Ericzen, Brookman-Frazee, & Stahmer, 2005; Bebko, Konstantareas & Springer, 1987). The complexities associated with raising a child with ASD put parents at greater risk, which highlight the importance of understanding potential stressors that impact parental wellbeing. While parenting stress is well documented among parents of children formally diagnosed with ASD, little is understood about families whose children are at-risk or newly diagnosed. The prevalence of ASD is now reported in 1 in every 88 children (CDC, 2012) in the United States, and with recent advancements in early screening and diagnostic measures, children are being diagnosed at much younger ages than previously possible. As a result, further inquiry into the potential stressors experienced among this group of parents is essential. Thus, the aim of this study is to examine stress profiles of parents of younger children at-risk, as well as the association of parenting stress and working alliance between caregivers and early intervention providers. Given the paucity of research on family wellbeing and young children at-risk, this study aims to broaden our understanding of parenting stress to a much younger age group of children at-risk.

Autism Spectrum Disorder is a neurodevelopmental disorder that affects cognitive, social, and behavioral functioning. ASD now encompasses related disorders, including autism, Aspergerâ€™s syndrome, and pervasive developmental disorder-not otherwise specified (PDD-NOS). Although autism symptoms are highly variable, there are unifying characteristics among children on the spectrum that include difficulties with social relatedness, communication, changes in routines, and repetitive behaviors. Onset of ASD symptoms must be present before 3 years of age, and formal diagnosis is possible using the following diagnostic framework: criteria from the Diagnostic and Statistical manual of the American Psychiatry Association (DSM-IV), algorithm scores from the Autism Diagnostic Observation Schedule (ADOS) (Lord, 2000), parent interviews, and medical history records. The ADOS is a standardized, semi-structured assessment and is considered the â€˜gold standardâ€™ in diagnosis of ASD. Previously, reliability and consistency of ADOS scores across development could be made by 3 years of age (Charman, Taylor, Drew Cockerill, Brown, & Baird, 2005). However, with recent advancements on early warning signs of ASD diagnosis can now be reliably made as early as 24 months (Lord, Luyster, Guthrie, & Pickles, 2012). ASD is now considered a significant public health concern with the prevalence increasing each year (Zwaigenbaum, Bryson, Lord, Rogers & Carter, 2009). The American Academy of Pediatrics now recommends developmental surveillance of all children, with ASD screening beginning at 9-months of age, and continuing at 18- and 24-months. Early parent concerns are generally expressed first to pediatricians, and commonly relate to their childâ€™s delay in speech and communication (Rogers, 2009). While a

â€˜wait and seeâ€™ attitude may have previously been the norm, developmental surveillance could potentially ameliorate this issue. Detecting early signs of ASD is essential for implementation of early intervention services during a critical period of development, which research has demonstrated improves family adjustment and child outcomes. While there are no reliable biological markers of ASD, studies have informed the early behavioral signs in very young children (Rogers, 2009). The first studies on early indicators of ASD are retrospective in nature, relying on parent report and videotapes of infants in their first year. According to parent report on children later diagnosed with ASD, concerns were raised by the second year, and one-third of parents report concerns within the first year (Ozonoff, Young, Steinfield, Hill & Cook, 2009; Baird, Charman, Pickles, Chandler, Loucas, 2008; Zwaigenbaum et al., 2009). Among the most common parental concerns were delays in language development, and lack of responsiveness, generally in response to the childâ€™s name being called (Zwaigenbaum et al., 2009). Parents also report that children appeared isolated, played differently from other children, and did not ask for help from an adult (Dahlgren & Gillberg, 1989). In 15-40% of children, parents report a regression, or a marked loss of language or acquired skills by the second year (Chakrabarti & Fombonne, 2001; Ozonoff, et al., Longitudinal data would be valuable to gain a deeper understanding of the process of raising a child at-risk, adjusting to diagnosis, and the decision to uptake intervention services. For the parents in this study who reported clinical levels of stress, the diagnostic transition could be a critical period and may require specific consideration within early intervention programs. Research suggests that family adaptation and adjustment to diagnosis is strongly correlated to social support and coping strategies, and alliance between treatment providers and parents may play a key role. As early intervention has become more family centered, focusing on the needs and strengths of the individual family system is paramount. Early interventionists may play a significant role in helping to facilitate the adaptation process. The relationship that forms between parents and service providers within early intervention settings could positively impact parental perceptions and adjustment over time, and reduce negative impacts on the family system.

The effectiveness of a strong alliance on parenting stress suggests the need to incorporate partnerships within early intervention settings. Increasing parent participation, goal setting, and collaboration could have positive results on parental perceptions, and could influence the way early intervention services are

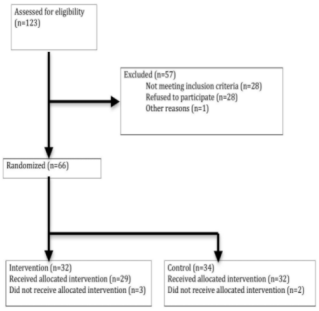

Figure 1. Participant Flow Chart

	Control N = 22	Treatment N = 23
Child Demographics		
Gender (%)		
Male	73.5%	84.4%
Female	26.5%	15.6%
Ethnicity (%)		
White	50.0%	34.8%
Hispanic	22.7%	26.1%
Other/Mixed	27.3%	39.1%
Parent Demographics		
Gender (%)		
Male	9.0%	0.0%
Female	91.0%	100%
Education (%)		
Less than HS	4.5%	4.3%

Figure 4: Generated HTML

8 Testing / Evaluation / Assessment

8.1 Common Object in Context Evaluator

To evaluate prediction results of the model, the Common Object in Context (COCO) Evaluator is selected to perform the task. The following 6 metrics are used to characterize the performance of object detectors on COCO:

1. Average Precision (AP)[2]:
   (a) AP (mAP) - Mean of Percentages of APs at IoUs from 0.50 to 0.95 (stepsize = 0.05)
      i. $IoU = \{0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$
      ii. (primary challenge metric)
      iii. The evaluator makes no distinction between AP and mAP
   (b) AP50 - Percentage AP at IoU = 0.50
      i. (PASCAL VOC metric)
   (c) AP75 - Percentage AP at IoU = 0.75
      i. (strict metric)
2. AP for Different Sized Objects[3]:
   (a) APs - Percentage AP for small objects: $\texttt{area}_{\texttt{bounding box}} < 32\texttt{pixels}^2$
   (b) APm - Percentage AP for medium objects: $32\texttt{pixels}^2 < \texttt{area}_{\texttt{bounding box}} < 96\texttt{pixels}^2$
   (c) APl - Percentage AP for large objects: $\texttt{area}_{\texttt{bounding box}} > 96\texttt{pixels}^2$

IoU threshold is a value used in object detection to measure the overlap of a predicted versus actual bounding box for an object. The closer the predicted bounding box values are to the actual bounding box values the greater the intersection, and the greater the IoU value, from 0.50 to 0.95.

The Average Precision (AP) lies within the range of $[0, 1]$ (represented as a percentage) and is defined by the average of the precision values, when plotted against recall, where precision measures the accuracy of the model's predictions and recall measures how well the model identifies labels correctly [2]. In mathematical terms, these definitions are defined as:

$$\texttt{Precision} = \frac{\texttt{True Positives}}{\texttt{True Positives + False Positives}}$$

$$\texttt{Recall} = \frac{\texttt{True Positives}}{\texttt{True Positives + False Negatives}}$$

In the evaluator, the precision (p) and recall (r) of each prediction is calculated. From here, the evaluator takes the average of these precision values to give the resulting AP.

In other words, precision answers what proportion of positive identifications were correct; conversely, recall answers what proportion of actual positives were identified correctly. As informed by the client, an AP greater than or equal to 0.75 is considered strong enough for industry standard usage.

[2]For AP calculation, the evaluator calculates precision and recall, and then generates a precision-recall function which is integrated across [0, 1] to return the final result [2].

[3]Here 32 and 96 refer to the number of pixels, the superscript means square pixels, and the comparison is with the area of the bounding box.

8.2 Results for Model Trained on 1 ETD

To prepare for testing and evaluation on larger datasets, an initial model was trained on 1 ETD to ensure no bugs and/or errors existed within the routine. After training, the model was given to the COCOEvaluator and the results in Table 3 were found.

Table 3: Results for Model Trained on 1 ETD

AP	AP50	AP75	APs	APm	APl
6.878	13.257	5.009	0.000	3.003	12.043

id	AP	id	AP	id	AP
elements	nan	abstract-text	nan	abstract-title	nan
author	nan	chapter-title	nan	committee	nan
contents	0.000	date	nan	degree	nan
footnote	0.000	link	nan	paragraph	41.266
reference	0.000	section	0.000	title	nan
toc-title	0.000	university	nan		

As expected, AP scores in part and on the whole were well short of the mark with every true positive for labels other than `paragraph` being misidentified. A score of `nan` indicates there are no true positives for the respective element; these scores merely state the label does not exist within the dataset. A score of 0, however, indicates the element **does** exist within the dataset, but the element was never identified in the model.

These results were expected on the grounds that the model was only trained on a single ETD. As with most machine learning models, model adequacy is directly correlated with the amount of information used to train the model. Thus, using a single ETD, an AP of 6.878 was achieved on average across all thresholds, due to a very confused model, and with `paragraph` reaching a sub-par AP of 41.266, even though there are many paragraphs present in an ETD.

8.3 Results for Models Trained on 50 ETDs

Once the training and evaluation routine was assured using a single ETD, additional annotation data was generated in RoboFlow [9] for 50 scanned and 50 born-digital ETDs. This data was then imported and two new models were trained (one for the scanned, and one for the digital). The results of these models were:

In contrast to the model trained using a single ETD, the two models trained on 50 ETDs yielded relatively higher AP scores across all thresholds (24.650 for the scanned model, 28.156 for the digital model). This serves as proof of concept that more information used in training generally increases the model's adequacy.

When comparing the individual elements for each of the 50 ETD models, it is clear that elements with greater density within the dataset achieve much higher scores than sparse elements[4]. Such a loss of precision is generally mitigated through repeat factor sampling to offset the uneven distribution of elements.

[4]For example, footnote, reference text, table, figure, and paragraph, yielded AP scores greater than 50; on the other hand, elements such as title, author, degree, date, and university – that generally only appear once or twice in the ETD – rarely achieved an accurate prediction

Table 4: Results for Model Trained on 50 Scanned ETDs

AP	AP50	AP75	APs	APm	APl
24.650	42.153	23.972	8.727	17.206	24.668

id	AP	id	AP	id	AP
chapter-title	21.479	equation	47.795	paragraph	64.080
chapter-subheading	19.727	equation-number	25.815	reference-heading	0.000
degree	0.000	figure	62.250	reference-text	76.897
title	0.000	figure-caption	39.812	supervisor	nan
abstract-heading	0.000	foot-note	60.158	table	69.630
abstract-text	0.000	list-of-content-heading	0.000	table-caption	27.985
author	0.000	list-of-content-text	0.000	university	0.000
date	0.000	page-number	26.680		

Table 5: Results for Model Trained on 50 Digital ETDs

AP	AP50	AP75	APs	APm	APl
28.156	48.292	26.993	16.719	22.563	40.958

id	AP	id	AP	id	AP
chapter-title	13.179	date	16.238	list-of-content-text	66.729
chapter-subheading	29.576	degree	0.000	page-number	21.300
title	0.000	equation	48.631	paragraph	74.533
abstract-heading	0.000	equation-number	21.645	reference-heading	16.365
abstract-text	0.000	figure	73.803	reference-text	80.223
algorithm	0.000	figure-caption	42.484	table	53.465
author	5.406	foot-note	60.387	table-caption	39.252
committee	0.000	list-of-content-heading	12.525	university	0.000

Finally, due to the higher clarity of born-digital ETDs when compared to scanned ETDs, text elements such as `paragraph` or `reference text` resulted in higher AP scores in the digital model. When scanning older ETDs, the level of detail can at times be adversely affected by the quality of the scanner as well as general wear-and-tear of the preserved document. However, alterations to the original document (i.e., smudging, tears, creases) are nonexistent when dealing with born-digital documents.

8.4 Results for Model Trained on 100 ETDs

To evaluate the performance of the model when trained upon both scanned **and** born-digital ETDs, the datasets used in the previous section were merged to create a new 100-ETD dataset. The merged model yielded the following results:

Table 6: Results for Model Trained on 100 Merged ETDs

AP	AP50	AP75	APs	APm	APl
21.727	38.328	21.255	15.147	15.765	21.649

id	AP	id	AP	id	AP
chapter-title	18.568	degree	0.000	paragraph	70.653
chapter-subheading	22.432	equation	39.679	reference-heading	0.000
title	0.000	equation-number	20.444	reference-text	63.894
abstract-heading	0.000	figure	65.616	supervisor	nan
abstract-text	0.000	figure-caption	41.944	table	67.552
algorithm	0.000	foot-note	51.508	table-caption	34.425
author	0.000	list-of-content-heading	0.000	university	0.000
committee	0.000	list-of-content-text	0.000		
date	1.188	page-number	23.538		

Contrary to expectations, the merged model achieved lower scores than the smaller, exclusive models. The merged model dropped its AP across all thresholds to 21.727, indicating a greater loss of precision. This loss of precision is mostly attributed to the subtle differences in quality and orientation between the two ETD types.

Scanned ETDs are generally older and date back to before the advent of digitally stored media (i.e., 70's and 80's). These documents would have been transcribed via a typewriter or printer, which means the quality of the document is dependent upon the levels of ink. Too much ink leads to smudging and bold/cramped text, whereas too little ink leads to lower opacity and loss of detail. Digital ETDs, however, have constant levels of detail throughout the document. As a result, merging scanned and digital ETDs into a single dataset does not improve the model.

8.5 Results for Models Trained after Repeat Factor Sampling

In an effort to get better results on sparse elements for the previously trained models, each of the datasets was trained using Repeat Factor (RF) Sampling. The results of the RF models were as follows:

Table 7: Results for Model Trained on 50 Scanned ETDs w/ RF Sampling

AP	AP50	AP75	APs	APm	APl
24.271	42.516	24.342	8.591	17.924	24.318

id	AP	id	AP	id	AP
chapter-title	21.318	equation	42.729	paragraph	67.561
chapter-subheading	14.374	equation-number	24.503	reference-heading	0.000
degree	0.000	figure	62.643	reference-text	64.978
title	0.000	figure-caption	42.014	supervisor	nan
abstract-heading	0.000	foot-note	58.881	table	59.509
abstract-text	0.000	list-of-content-heading	0.000	table-caption	36.164
author	0.000	list-of-content-text	9.901	university	0.000
date	0.000	page-number	26.032		

Table 8: Results for Model Trained on 50 Digital ETDs w/ RF Sampling

AP	AP50	AP75	APs	APm	APl
27.910	49.091	27.576	14.563	22.905	39.634

id	AP	id	AP	id	AP
chapter-title	13.440	date	10.495	list-of-content-text	59.982
chapter-subheading	30.032	degree	0.000	page-number	21.206
title	0.000	equation	48.243	paragraph	72.445
abstract-heading	0.000	equation-number	21.515	reference-heading	10.461
abstract-text	0.000	figure	69.651	reference-text	83.272
algorithm	0.000	figure-caption	42.948	table	61.366
author	8.581	foot-note	58.569	table-caption	42.702
committee	0.000	list-of-content-heading	14.930	university	0.000

After analyzing these results, minor fluctuations in precision were present for sparse elements. Elements originally yielding an AP of 0 generally continued to yield the same result after RF sampling and were not improved upon. At times elements with weak AP scores less than 50 showed minor improvement in some models; but, these improvements were not consistent between all three models[5].

Changes for AP across thresholds and scales were also minor and inconsistent. It follows that RF sampling, while a valuable tool in object detection modeling, did not necessarily improve the model's adequacy.

[5]For example, chapter title improved for the scanned and digital datasets but was reduced in the merged dataset; chapter subheading improved for the digital and merged dataset but was reduced in the scanned dataset.

Table 9: Results for Model Trained on 100 Merged ETDs w/ RF Sampling

AP	AP50	AP75	APs	APm	APl
22.383	38.876	21.867	10.026	15.186	22.316

id	AP	id	AP	id	AP
chapter-title	16.366	degree	0.000	paragraph	69.413
chapter-subheading	24.229	equation	43.083	reference-heading	2.376
title	0.000	equation-number	17.197	reference-text	81.635
abstract-heading	0.000	figure	65.005	supervisor	nan
abstract-text	0.000	figure-caption	39.481	table	61.285
algorithm	0.000	foot-note	58.096	table-caption	33.356
author	2.030	list-of-content-heading	0.000	university	0.000
committee	0.000	list-of-content-text	0.000		
date	0.000	page-number	23.648		

However, given larger datasets and samples in future development, the RF sampling technique may prove beneficial.

9 Users' Manual

9.1 Setting Up the Environment on Windows

In order to successfully run and use the pipeline, the user can choose to run the provided script directly in Google Colab or on a Linux server. If the user wants to run it locally on a Windows machine, they need to set up their environment with the following steps.

Step 1: Prerequisites

First, Anaconda [1] and CUDA 10.2 [7] need to be installed. Next, PyTorch should be installed. Due to compatibility issues with PyTorch and CUDA, make sure to select the right version. To check the version compatibility, use the PyTorch Version Web Page [8].

After Anaconda, CUDA, and PyTorch are installed, install Cython and Pycocotools. Use the following code in the Anaconda prompt to install Cython.

```
pip install cython
```

Additionally, use the following code in anaconda prompt to install Pycocotools.

```
pip install "git+https://github.com/philferriere/cocoapi.git#egg=pycocotools&subdirectory=
                                                PythonAPI"
```

Step 2: Install Detectron2

The Detectron2 files need to be cloned from the Facebook AI Research open sourced repository [11]. Then, in the Anaconda prompt, go to the cloned repository and install Detectron2.

```
python -m pip install -e detectron2
```

Step 3: All Other Required Packages

Run each line of code below in the Anaconda prompt to install the necessary packages:

```
#for pdf to image convertion
pip install pdf2image
pip install pyyaml==5.1
#For loading data from RoboFlow
pip install roboflow
#the following 2 are for OCR
pip install pytesseract
pipe install pymupdf
```

9.2 Setting Up the Environment on Linux

Step 1: Prerequisites

On Linux, Python, along with its corresponding `pip` command, can be used to install all the necessary packages. It is advised to use a new Python virtual environment to make sure there are no package conflicts. A Python version of `3.7` or greater is needed for all the required packages.

Step 2: PyTorch and Detectron2

Once the virtual environment is activated, the approprate packages can be installed. Detectron2 can only work with certain PyTorch and CUDA versions, so, before installing PyTorch, it is important to determine the CUDA version of the GPU and download an appropriate version of PyTorch. The bash command `nvidia-smi` can be used to see the current version of CUDA on the GPU. Next, check the Install Detectron2 web page for the most recent compatible versions of PyTorch and the CUDA version of the GPU, and, before installing Detectron2, install the appropriate version of PyTorch [12, 8]. After PyTorch is installed, the command for installing Detectron2 can once again be found from the Install Detectron2 web page.

Step 3: All Other Required Packages

Now that Detectron2 and PyTorch are set up, other packages for PDF to image conversion, OCR, and PDF manipulation can be installed with the following commands:

```
pip install pdf2image
pip install pytesseract
pip install pymupdf
```

9.3 Running the PDF to XML Pipeline

Now that the environment is set up, the ETD's PDF is ready to be converted to XML. The PDF should be on the same machine as the pipeline. All of the required functions for conversion are located in `pdf_to_text.py`, and should be used in the following order:

1. Before starting the pipeline, the list of categories that the object detection model was trained on needs to be loaded in as a Python list. The list of categories can be found from the model's corresponding dataset folder, as described in Listing 2.
2. Next, the trained model should be loaded in with the `get_predictor` function. This function takes in the path to the model weights, which is stored with the PyTorch `.pth` file extension, and the number of categories the model predicts (this should be the length of the list obtained in Step 1).
3. The output of `get_predictor` is the configured object detection model, which is now ready to make predictions. This configured model should be passed into the `predict_annotations` function, along with the list of categories and the path to the PDF file.
4. Now that the predictions have been made from the PDF, either `parse_scanned` or `parse_digital` should be used to parse the text and images. Which function to use depends on whether the PDF is

born-digital or scanned. If it is born-digital, `parse_scanned` can still be used, but, because it uses OCR to extract the text rather than extracting from the original PDF file, the results will not be as accurate.

5. Now that the text and images have been parsed from the PDF, the `create_xml` function can be used to format the results into XML.

10 Developer's Manual

10.1 Dataset Access

The created dataset is on RoboFlow, and, as such, a RoboFlow account is needed to access the dataset. The dataset is separated into several small batches. Each batch contains 10 or 20 ETDs. ETDs with identifier from 0 to 100 are scanned ETDs, while ETDs with identifier from 900 to 1000 are born-digital ETDs. If a combination of multiple batches is needed, use the merge function on RoboFlow as shown in Figure 5.

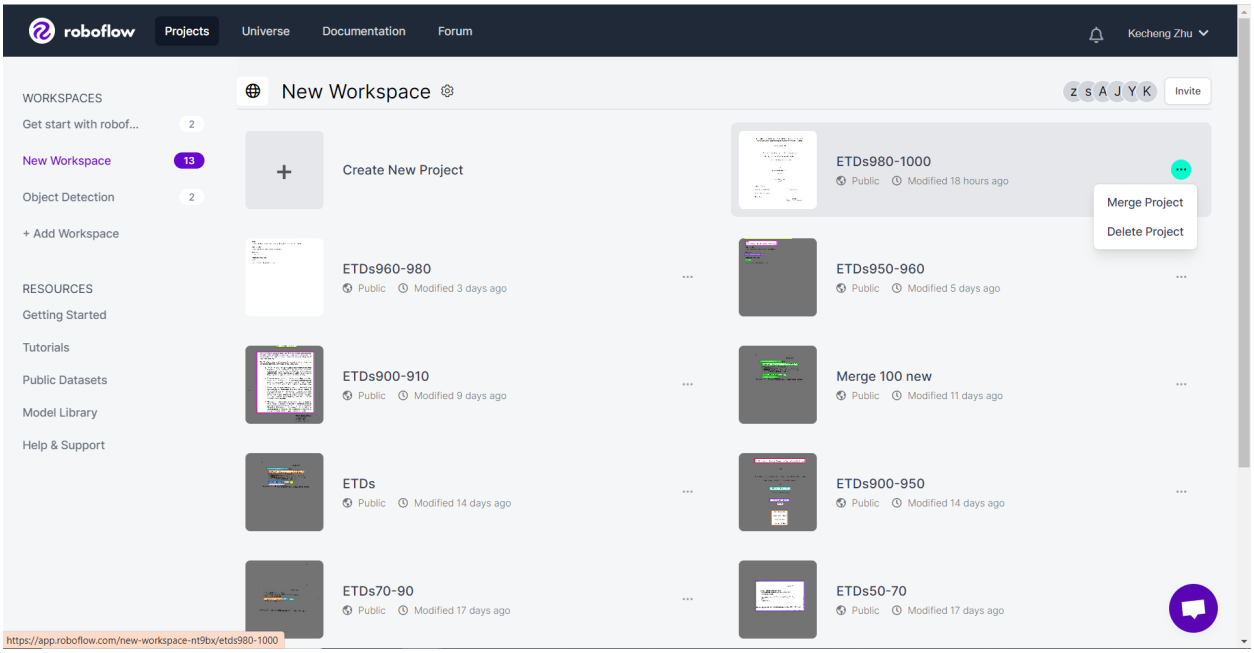

Figure 5: RoboFlow Merge Data Sets

In order to access the data in each project, go to the version page in a project. Choose any existing version or click on generate new version to modify image resize, data set augmentation, and class remapping as shown in Figure 6.

After generated a data set, click on export to use the generated data set. Directly download the whole data set into a zip file and use the download code provided by RoboFlow. For download code, use the following code segment to load the data set.

```
rf = Roboflow(api_key="<User API KEY GENERATED BY ROBOFLOW>")
project = rf.workspace("<Name of work space>").project("<Name of project in workspace>")
dataset = project.version(<Version ID>).download("<Data set format>")
```

After the dataset is downloaded, the dataset should be structured in the file heirarchy described by Listing 2.

Code Listing 2: Dataset Folder Hierarchy

```
<dataset_name>
```

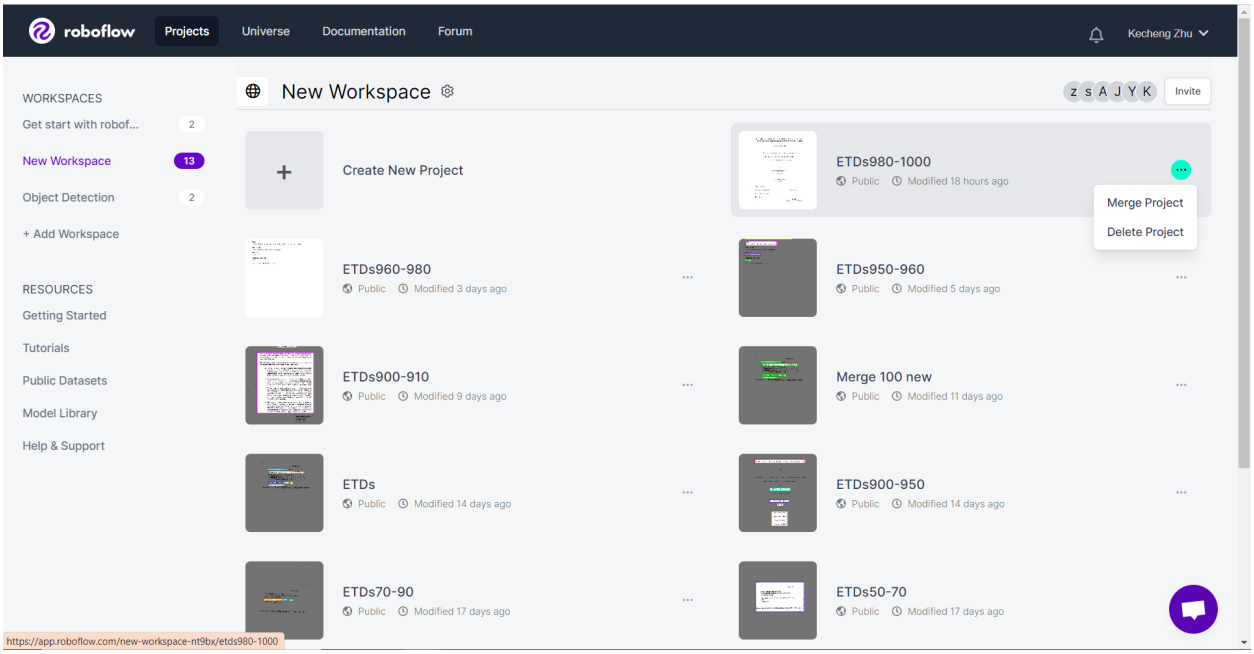

Figure 6: Generate Data Set in RoboFlow

```
|— train
    |— <img1>.jpg
    |— ...
    |— _annotations.coco.json
|— test
    |— <img1>.jpg
    |— ...
    |— _annotations.coco.json
|— valid
    |— <img1>.jpg
    |— ...
    |— _annotations.coco.json
```

10.2 Training / Evaluating / Validating the Model

A number of Python functions were created for manipulating the dataset, training and evaluating the Detectron2 model, and validating its results by comparing the actual PDF with the generated HTML.

It is important to verify the dataset before training and to check that the label categories are correct. For example, even after mapping the categories through the RoboFlow UI, many unnecessary and unused categories were still in the `_annotations.coco.json` files, such as `pag-`, `eq-`, and `Chapter subheading-` (when it should be `Chapter subheading`). The categories, images, and annotations in a dataset can all be manipulated and visualized through the `_annotations.coco.json` files found in each `train`, `test`, and `valid` folder. A number of functions have been defined in `clean_coco.py` that can be used for handling the dataset.

- The `visualize_image` function can be used to visualize an image by its image ID from a dataset. This function uses the annotations associated with the image ID to draw bounding boxes around each human annotated element, along with the corresponding label for the type of element.
- The `map_categories` function can be used to label one category as another existing category. For example, if called with the categories `(Chapter subheading-, Chapter subheading)`, all objects that were labeled as `Chapter subheading-` will now be labeled as `Chapter subheading`, and the `Chapter subheading-` category will be deleted.
- The `delete_pages_with_category` function can be used to delete a specific category, like `eq-`, and delete all the images that had an annotation with that category. It is important to check the number of pages with that category before deleting them from the dataset - this can be done with the `find_images_with_category_id` function.

After the dataset has been cleaned and visually verified, the Detectron2 model is ready to be trained. This can be done using the `train.py` script. This script is meant to be run from the command line with the command `python train.py <DATASET_FOLDER> <MODEL_OUTPUT_DIRECTORY>`. The dataset folder should be a path to a dataset with the hierarchy described in Listing 2, and the resulting `.pth` model weights will be stored in the model output directory. Additionally, the `--repeat_factor` flag can be passed to indicate training with repeat factor sampling, as discussed in Section 7.3.1. This script trains on on the data found in the dataset folder's `train` directory.

After the model is trained, it should be evaluated using the `eval_to_csv.py` script. This script evaluates the model's performance based on the dataset folder's `test` directory, and stores the evaluation metrics in a CSV file. Lastly, after the evaluation metrics are obtained, the model should be visually validated against real images. This can be done using the `visualize_results` function found in `model_utils.py`. The results should be visualized from the images in the `valid` directory in the dataset folder.

10.3 PDF to XML Pipeline

After the model is trained, it can now be used in the PDF to XML pipeline. Much of the pipeline is described in Section 9.3. For testing purposes, the function `obtain_annotations` can be used to retrieve the human-labeled annotations for a specific PDF. This function takes in a dataset folder, as described in Listing 2, and a path to the PDF. This function can also simultaneously visualize the human-labeled annotations for the entire PDF. After using the function, the human-labeled annotations can then be used in the rest of the XML pipeline, rather than using a model's predicted annotations. This can be beneficial for testing the OCR in the `parse_scanned` function, the PDF parsing methods in the `parse_digital` function, and the XML rules in the `create_xml` function. The XML rules are described in Section 7.5.

10.4 XML to HTML Script

In order to visualize the XML file in a human readable format, the function `XML2HTML` could be used. This function has a parameter `in_dir`. This parameter allows the user to specify the directory where the output XML and images are located. If this parameter is not specified, the current directory is used as the default. This script assumes that the XML file is UTF-8 encoded and the XML tree follows the format stated in Section 7.5. That is for the first layer. The XML indicates different parts of the ETDs, including front, body,

and back. The second layer includes Title, Author, university, degree, committee, date, abstract heading, abstract text, table of content heading, table of content text, chapter, reference heading, and reference text. Chapter contains the third layer, and includes paragraphs, figures, tables, equations, algorithms, footnotes, and subheadings.

11 Lessons Learned

11.1 Timeline

The timeline of the project is shown in Table 10.

Date	Event
2.4	Finish reviewing requirements and meet with client
2.11	Convert ETDs into pictures/Working on presentation
2.18	Pres 1
2.25	Get started with labeling/Model selection
3.4	15% Labelling work
3.18	30% Labelling work/Train with Labeled dataset
3.25	50% Labelling work
4.1	65% Labelling work/Start OCR part
4.8	75%/ Labelling work/Retrain model/Finish OCR part
4.14	85% Labelling work/Save results to XML part/Interim report
4.15	Pres 2
4.22	100% Labelling work/Convert XML to HTML/Train with 200 ETDs
5.1	Final report/Final pres

Table 10: Timeline

11.2 Problems / Solutions

11.2.1 Annotations in RoboFlow

For first-time users of RoboFlow, annotating documents can feel a bit cumbersome. Labeling thousands of images for an extended block of time gets tedious; thus, it is easy to get distracted if multitasking while labeling. It is important for the human annotators to be aware of some rules which will help prevent human error and reduce later revision.

1. Projects in RoboFlow **cannot** be exported unless the project is in a **public** workspace. To prevent the possibility of annotating without the ability to export the dataset later, the originator of the workspace **must** create a **public** workspace. If a private workspace is created, the workspace cannot later be changed to public.
2. Annotations in RoboFlow are case-sensitive. For example, if one person in the group labels each paragraph as `paragraph` but another person in the group labels them as `Paragraph`, these labels will

be recorded differently. To prevent duplicated class names for the same label, each annotator in the group should determine an appropriate format (i.e., UpperCamelCase, lowerCamelCase, underscores) to maintain throughout the labeling process.

3. After merging two datasets into one, revising a label in the merged dataset will not affect the label in the dataset it originated from. For example, if two people in the group label their own chunks of ETDs and then merge these projects but an annotation is found to be missing or incorrect afterwards, fixing the issue in the merged project will not fix the issue in the pre-merged project. Should an issue be found later in the process, it would be easier to fix the annotation(s) in the original project(s) and **then** re-merge those projects. This way, should someone in the group want to use any of the RoboFlow projects in the shared work space they will always be accurate.

4. When annotating a blank page, RoboFlow has an option to mark the page as NULL. Until the page is labeled or marked NULL, the dataset cannot be generated in COCO format. When dealing with this situation, the annotator should always mark the page as NULL rather than introduce new prediction classes (i.e., `blank page`). It should be noted that blank pages that have page numbers should still receive an annotation for `page number`.

11.2.2 RGB vs. BGR

For the traditional RGB three-color map, it is actually a three-channel (R, G, B), and each channel is represented by an 8-bit unsigned number (0-255 colors). Additionally, a three-channel image usually represent a single image in the RGB order. Our dataset and model are all based on the RGB order. In `cv2`, the three channels are arranged according to BGR, that is, arranged in the order of blue, green, red.

For Detectron2's `Visualizer` tool, the function takes the image array in RGB format; thus, the array provided by `cv2`'s `imread()` function, which reads a image from a specified file, must be converted prior to the `Visualizer` function call. This can be accomplished by appending `[:, :, ::-1]` to the BGR array. This matrix operation should also be performed prior to any calls in `pyplot` from `matplotlib`, which also uses RGB format.

11.2.3 Version Control

The code used to generate training, evaluation, and visualization scripts requires the various libraries, packages, and environments being used to remain interoperable. Changes or updates to code happen frequently and old functions often become deprecated when new methods are adopted. Accordingly, there were some issues at first with getting the libraries and packages to work with the environment. This was solved by installing a new virtual environment using Python 3.7. Currently, the scripts require this version of Python to run seamlessly. Future developers should continue to monitor any changes to these libraries and packages which could disrupt this seamless interaction.

11.2.4 Remote Access

Training a model over thousands of iterations can require extensive amounts of compute time. Since these models are being trained remotely on a private server, a loss of connection can potentially interrupt any

processes being run remotely.

To prevent training interruptions due to connection loss, the developer should always invoke the `screen` command. A screen is a terminal multiplexer, meaning the developer can start a screen session and run processes that persist until completion, error, or the screen receives a termination signal from the developer.

It is recommended that each developer start their own screen session which can then be detached and reattached as necessary while training models or running other scripts. To exit or terminate a screen completely, simply invoke `exit` or send a `SIGTERM` signal.

11.3 Future Work

There are several areas for improvement. One of the most important areas of improvement is dataset enrichment. For now, the size of the dataset is still quite small compared to similar work done by others. Moreover, the dataset is highly imbalanced. This imbalance is unavoidable when labeling one ETD at a time, since only one or two occurrences of the title will appear in an ETD, while there will be hundreds of paragraphs. One way to address this imbalance is to simply label more ETDs, which will eventually increase the occurrence of infrequent labels to where the model can successfully learn the task (this usually requires at least 1500 occurrences of each element in the dataset based on the dataset health check provided by RoboFlow). This would improve the accuracy of the model, and the subsequent OCR and XML pipeline would be more meaningful. But labeling is a labor-intensive task, so relying on existing trained models for labeling may be a reliable and more cost-effective approach. To do this, the trained model would be the first to create the annotations, and then the model's annotations would be imported into the annotation tool, where they can be manually verified and updated. With this method, human annotators can focus less on labels that the model can already recognize well, such as paragraphs, pictures, and tables, and more on labels that are less frequent. Manual annotation not only annotates the elements that have not been recognized, but also corrects the annotation of the machine to ensure the correctness of the data set.

Another important area of improvement is tuning the hyper-parameters of the model during training. Tuning a model's hyper-parameters, like learning rate, batch size, and weight decay, can improve a model's training time and performance [10]. Currently, the model trained with a constant learning rate of 0.00025, with a max iteration of 10000, and with 2 images per batch. Additionally, repeat factor sampling for addressing class imbalance showed promise, as discussed in Section 8.5 – however, only one level of the hyper-parameter `REPEAT_THRESHOLD` was tested. After training, the model results could be improved by adding post-processing rules for the model's predictions, such as limiting the number of page number predictions to one per page.

Lastly, the rules for creating the XML document could be improved, namely the rules for associating elements with each other. For example, it is difficult to define when a figure corresponds with a figure caption. In the current pipeline, a figure is associated with a figure caption if the figure caption is the last or next detected element in terms of the y-coordinate. However, this is not 100% accurate, as another element like a page number could be in-between the two.

12 Acknowledgements

Aman Ahuja - Ph.D. student in the Department of Computer Science. Email: aahuja@cs.vt.edu

Edward A. Fox - Professor of Computer Science, and, by courtesy, in the Dept. of Electrical and Computer Engineering, Virginia Tech. Email: fox@vt.edu

13 References

[1] ANACONDA. ANACONDA. https://www.anaconda.com/, 2022.

[2] Gad, A. F. Evaluating Object Detection Models Using Mean Average Precision (mAP). https://blog.paperspace.com/mean-average-precision/, 2020.

[3] Gupta, A., Dollár, P., and Girshick, R. LVIS: A Dataset for Large Vocabulary Instance Segmentation, 2019.

[4] Li, M., Xu, Y., Cui, L., Huang, S., Wei, F., Li, Z., and Zhou, M. DocBank: A Benchmark Dataset for Document Layout Analysis, 2020.

[5] Li, Y., Wang, T., Kang, B., Tang, S., Wang, C., Li, J., and Feng, J. Overcoming classifier imbalance for long-tail object detection with balanced group softmax. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 10988–10997.

[6] Lin, T.-Y., Patterson, G., and Ronchi, M. R. e. a. COCO Data Format. https://cocodataset.org/#format-data, 2014.

[7] NVIDIA. CUDA 10.2. https://developer.nvidia.com/cuda-10.2-download-archive, 2022.

[8] Paszke, A., Gross, S., Massa, F., and et al. Install PyTorch Web Page. https://pytorch.org/get-started/locally/, 2019.

[9] RoboFlow. RoboFlow. https://roboflow.com/, 2022.

[10] Smith, L. N. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. https://doi.org/10.48550/arxiv.1803.09820, 2018.

[11] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[12] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R. Install Detectron2 Web Page. https://detectron2.readthedocs.io/en/latest/tutorials/install.html, 2019.

[13] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. A comprehensive survey on transfer learning. https://arxiv.org/abs/1911.02685, 2019.