Object Detection

by

Annie Tran

Andrew Leavitt

Brian Dinh

Kevin Dinh

CS 4624: Multimedia, Hypertext, and Information Access

Virginia Tech

Blacksburg, VA 24061

Dec. 15, 2022

Instructor: Dr. Edward A. Fox

Client: Aman Ahuja

Table of Contents

List of Figures

List of Tables

Executive Summary

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 and represented by 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 summarization.

The goal of this project is 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 represent them with an XML document. A pipeline to convert a dataset and label it into these different elements was developed.

As a past iteration of this project, the Spring 2022 group has annotated 200 ETDs, digital and scanned, using an online tool called Roboflow. A model based on Facebook's open-sourced object detection model, Detectron 2, was trained using the created dataset. However, due to limitations with Roboflow, we had to create a new pipeline that would convert ETD into PDFs where we would then perform AI-aided annotations using their trained model.

Over the Fall 2022 semester, 20,000 image samples from ETDs were annotated using a Python package called PyLabel. PyLabel is an open-source Python library used to label PDFs. PyLabel can also take a trained dataset and use it for AI-aided annotations. A pipeline was created in order to divide the dataset into equal pieces, where a user can select the number of samples they want to annotate. Then old sample data is cleared out and replaced with new sample data that contains classes with low accuracy. Finally, the object detection team annotates and saves the annotations as a YOLOv7 .txt file which is accumulated in order to retrain the model with 10,000 annotated images and finally with 20,000 annotated pages.

1 Introduction

1.1 Background

1.1.1 Electronic Theses and Dissertation

Electronic theses and dissertations (ETDs) are the digital representations of theses or dissertations that represent the work of a single student to partially fulfill their Masters or Ph.D. requirements. These works can contain content from Math, Science, Technology, and the Humanities. These are important pieces of research information for both the University and for future researchers in the field.

ETD formats vary from University to University and do not follow a single standard. However, most ETDs contain the same classes, such as metadata (title, author, degree, year), chapter heading and subheadings, equations, figures and tables (with captions), and paragraphs. Many of the ETDs looked at will be single-columned.

In this project, there will be two types of ETDs, scanned and born-digital. Scanned ETDs take an existing physical ETD and convert it into digital. Born-digital ETDs, on the other hand, originate in digital form. Their text and information are embedded in a PDF file. This difference between scanned and digital will be later observed.

1.1.2 PyLabel and YOLOv7

The Fall 2022 object detection team has migrated from using Roboflow [1] and Detectron2 [2]. Instead, the team has opted to use the PyLabel annotation tool [3] on Jupyter Notebooks [4] and the YOLOv7 data format. PyLabel is a Python package that is able to convert datasets into another format (for example, YOLO to VOC or COCO), analyze annotations, split datasets, and label images using Jupyter Notebooks. PyLabel helps in preparing image datasets for object detection models such as PyTorch and YOLO datasets.

You Only Look Once version 7 (YOLOv7) [5] is an object detection algorithm that is known for its fast real-time object detection. In this project, the team solely uses this format for annotations, visualizations, and model training. A YAML file identifies all the classes used for annotation. This can be seen in Figure 1, where all the classes used during data labeling is identified in a YAML file. For each annotation, the bounding box and its corresponding class ID (defined in the YAML file) are recorded and saved in the dataset as Image and Label. PyLabel current support is for YOLOv5, however, since YOLOv5 and YOLOv7 dataset formats are the same, conversion between types is simple and the annotation tool was fitted to use the YOLOv7 file.

```
names:
- Metadata
- Chapter Title
- Chapter subheading
- Title
- abstract heading
- abstract text
- algorithm
- author
- committee
- date
- degree
- equation
- equation number
- figure
- figure caption
- foot note
- list of content heading
- list of content text
- page number
- paragraph
- reference heading
- reference text
- table
- table caption
- university
nc: 25
path: ..
train: train
val: val
```

Figure 1: Classes identified in YAML data format for Object Detection

1.2 Objective

The objective of this project is to create and train an object detection model to extract various elements of electronic theses and dissertations (ETDs) such as the metadata (document title, author, year), different headings, equations, figures & tables (as well as their captions), paragraphs, etc., and then represent them in an XML document. This project was started in Spring 2022 by a team of students from the CS4624 course, along with the client, and they were able to develop an object detection dataset to extract important elements from ETDs. The developed dataset was then used to train machine learning models such as faster-RCNN and YOLOv7. But, due to the numerical imbalance between each element within the document, the model performance differs across the classes and certain classes have lower accuracy/prediction rates than other classes. The Fall 2022 team is currently working towards improving the

performance of the classes that have lower accuracy values and also executing an experiment to test the efficiency of the model.

1.3 Deliverable

The purpose of this project is to improve and generate a dataset efficient in annotating important elements in ETDs primarily focusing on low-performing categories like metadata, equations, algorithms, etc. We aim to do this while completing other objectives as stated above, and while presenting the deliverables listed below:

- A dataset containing all annotations done in YOLOv7 format
- A figure representing the pipeline of the workflow it takes from converting an ETD into fully annotated images that will be saved in our dataset.
- Tables and figures representing the time it takes for AI-aided annotations vs. manual annotations
- A figure representing the results of the evaluation of our dataset
- Python code used throughout the semester represented in Google Colab

1.4 Team acknowledgments

Our team is composed of Brian Dinh, Kevin Dinh, Annie Tran, and Andrew Leavitt. Our team consists of senior (year 4) students majoring in computer science.

2 Requirements

2.1 Object Detection Pipeline

To convert an ETD into a format where elements/classes of the document can be identified using PyLabel and then saved into our dataset, a pipeline is needed. First, it is necessary to convert the ETD PDFs into images as an intermediate format so that the visually based Machine Learning model can identify and classify different elements and extract them.

Classes
- Metadata - Chapter Title - Chapter subheading - Title - abstract heading - abstract text - algorithm - author - committee - date - degree - equation - equation number - figure - figure caption - foot note - list of content heading - list of content text - page number - paragraph - reference heading - reference text - table - table caption - university

Table 1: List of all ETD Annotation Classes and Highlighted Minority ETD Annotation Classes

As stated in Section **1.2 Objectives** we have minority classes that will be the focus of our annotations which can be seen on the highlighted text in Table 1. These classes were selected as our minority classes under the instruction of our client of having a cutoff threshold of 90 for AP@0.5 for classes Table 2, and 500 # Instances for Table 3. Any class that fell under the threshold was classified by the team to be minority classes.

Category	AP@0.5	Category	AP@0.5
Title	92.5	Paragraph	97.4
Author	89.5	Figure	98.4
Date	68.3	Fig. Caption	95.4
University	91.1	Table	94.7
Committee	96.5	Tab. Caption	89.8
Degree	68.3	Equation	72.6
Abs. Heading	94.2	Eqn. Number	55.0
Abs. Text	86.7	Algorithm	66.6
LOC Heading	75.5	Footnote	98.9
LOC Text	99.3	Page Number	51.3
Chapter Title	88.8	Ref. Heading	80.7
Section	90.9	Ref. Text	99.3

Table 2: Model Training Results from Spring 2022

Category	# Instances
Title	439
Author	404
Date	338
University	309
Committee	282
Degree	279
Abstract Heading	169
Abstract Text	183
List of Contents Heading	512
List of Contents Text	1059
Chapter Title	2211
Section	9337
Paragraph	30359
Figure	6359
Figure Caption	5722
Table	2654
Table Caption	2213
Equation	5092
Equation Number	3051
Algorithm	96
Footnote	5722
Page Number	24543
Reference Heading	313
Reference Text	2088

Table 3: Total Number of Label Instances Per Class Labeled in Spring 2022

As our focus is to improve the performance of the minority classes we have to filter the classes and separate the images so that the machine learning program will only display and annotate

images that have the minority classes. Using this filter, we will be able to obtain more samples for these minority classes. These AI-annotated images will be reviewed by our team and exported into an image folder, as well as a YOLOv7 file describing the bounding boxes the classes in the corresponding images have.

2.2 Dataset Development and Pipeline process

The set of 500,000 ETDs, including both born-digital and scanned PDFs, is provided by the client. For the dataset to be processed by the team, the pages of all the PDFs need to be split. The dataset of ETDs was randomly split into 4 datasets of equal size which is used by a corresponding team member. At the team member's discretion, a batch of ETDs (samples) will be taken, converted into images, filtered to see only images with minority classes, go through AI-aided annotation, and then be exported into a training folder. This folder of the datasets will then be merged with the 'old' dataset completed in Spring 2022, and will go through evaluation which will welcome any improvements on the minority classes.

3 Methodology

This section will define the goals and subgoals that were used during the Fall 2022 semester for Object Detection and tasks done to achieve such goals.

3.1 Goals

Object Detection is the ability to use a Machine Learning (ML) model to detect different objects in an image. In our case, it is the ability of our ML model to detect different classes in an ETD. In the Spring of 2022, our predecessors manually annotated 200 total ETDs to develop and train a model to detect objects in ETDs. Our goal is to work with their dataset and, using AI-aided annotation tools, to improve the dataset on objects that the model has trouble working with.

Our goals:

- Annotate 20,000 pages containing minority classes from ETDs
- Retrain AI Prediction Model
    - After 10,000 annotated pages
    - After 20,000 annotated pages
- Time ourselves annotating with all versions of models to measure improvement

3.2 Tasks and Subtasks

1. Background tasks
    a. Research and understanding of predecessor's work
    b. Research and understanding of Machine Learning and associated Python libraries
        i. PyTorch
        ii. PyLabel
        iii. PDF2Images
        iv. YOLOv5, YOLOv7, and COCO formats
    c. Continuous documentation

2. Object Detection
    a. Split data set
        i. Receive dataset of ETDs from client
        ii. Split data randomly into 4 groups, and associate each split of the dataset with a team member
        iii. Save the divided data's file paths under the pickle Python object serialization library
    b. Annotate minority classes

i. Convert PDFs into images
ii. Filter images that detect minority classes using PyLabel and the old model
iii. Label converted images using PyLabel and the old model
iv. Validate all images have been annotated
v. Validate all classes of newly annotated images
vi. Export images and YOLOv7 data into a new dataset

3. Training and evaluation of new model
    a. Split new dataset into training and validation folders in 85:15 split
    b. Evaluate old model weight's performance on validation folder
    c. Using old model weights, train 100 epochs to create a new YOLOv7 weight
    d. Evaluate new model performance on validation folder
4. Side task
    a. Evaluation of Manually annotating vs. AI-aided annotation
        i. As an experiment of this project, we are tasked to observe the time differences between manual annotation
            1. Manually clicking a class label and creating bounding boxes around the class
        ii. and AI-aided annotations:
            1. Letting the model predict the image label first and then going in to fix what the model messed up

3.3 Description of Services

Service ID	Service Name	Input file name(s)	Output file name	Libraries; Environments	Description of Service
0	Splitting PDF repository	ETD repository	*.obj	glob pickle random	a *.obj file is created that will store the array that holds the file path of the ETDs created by glob, a module that returns all file paths that match a specific pattern, and random. Pickle is used so the divided ETD repository is not lost when the server/environment crashes. Division

					of repository ensures that separate users do not annotate the same ETD.
1	Selection of ETD to sample	*.obj	array of PDFs	Python	Returns *n* number of samples randomly selected from the split PDF repo.
2	Selection of Model	file path to epoch_#.pt	N/A	YOLOv7 PyTorch	User selects a model trained using YOLOv7. PyLabel service will use it to predict annotations.
3	Convert PDFs to Images	*.pdf	*.jpg	pdf2image	Convert PDF files into images that PyLabel service can use to annotate images using selected model
4	Analyze and label predicted images	N/A	N/A	PyLabel	Use the PyLabel labeling tool to predict ETD images.
5	Image export	*.jpg	training/images/*.jpg training/labels/*.txt	PyLabel	Export images to dataset
6	Split dataset into training and val	dataset file path	split_data/train/ split_data/val/	split-folders	Using split-folders, a user can determine the ratio to split the dataset into a training and val to be used for model training and evaluation.

7	Evaluate control (Old) model	dataset.yaml *.pt val/	eval_old.txt	YOLOv7 Google Colab	Using YOLOv7's testing code on Google Collab, we evaluated our control model using our testing datasets. It should result in a text based table telling us our results.
8	Model Training: control + new dataset	dataset.yaml *.pt		YOLOv7 Google Colab	Using the control dataset from last semester and the new dataset that we team members worked on the last few months. Use that dataset to train new generations of the models in the form of epochs on Google Colab.
9	Evaluate control + new dataset	dataset.yaml *.pt val/	eval_new.txt	YOLOv7 Google Colab	Evaluation of the old model using the control + new dataset on Google Colab. It should result in a text-based table telling our results.
10	Annotation Visualizer	dataset.yaml dataset file path	output.pdf	PyLabel	A user can visualize annotations they have done using PyLabel. Randomly select images from the new image dataset, and display random images for the group to understand and correct.

Table 4: Description of Services

3.4 Pipelines

The pipeline diagram in Figure 2 provides an outline of the main services from Service 3.3. The pipeline takes the ETD Dataset provided from our client, and randomly splits it into 4 equal datasets, for each of the project members, as seen in Service 0. This randomly split dataset will be used to provide the PDFs for the annotation process. The notebook will be cleared of old samples before we populate it with new samples from our dataset. In Service 3, we use the Python library pdf2images to convert our PDFs into annotatable images. In Service 4, we analyze and annotate the images using the prediction from the latest model at the time. We verify the data, and in Service 5 we export the annotated images and the containing data into a YOLOv7 format.

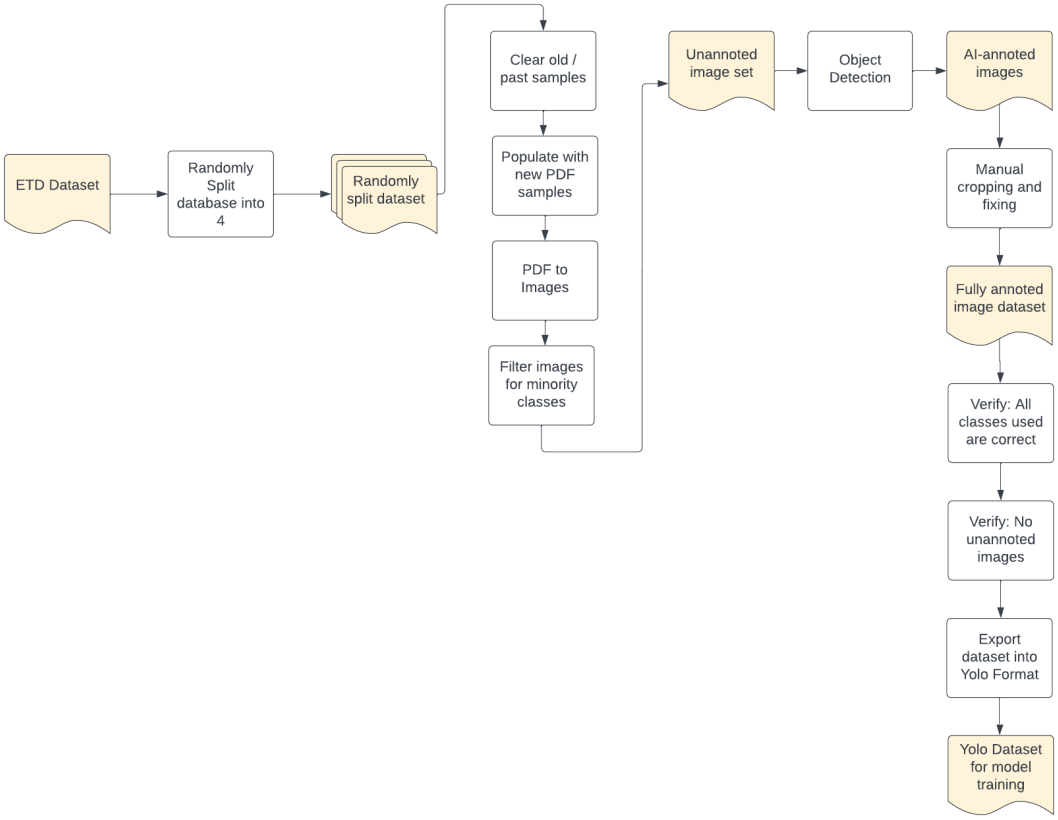

Figure 2: Annotation Pipeline Diagram

The pipeline diagram in Figure 3 provides an outline for the later services in Service 3.3. Before we can train a new model using the YOLOv7 dataset from Service 5, we first have to split the dataset. In Service 6, the YOLOv7 Dataset was split into 85% used for training the model, and 15% used for validation and testing of the model. Then we have to download both the split dataset, along with the old dataset, from SSH to local, for use in the model evaluation. In Service 7, the old dataset created from the Object Detection group of Spring 2022 was then evaluated as a control. In Service 8, we train and evaluate the new dataset that the team had worked on. Then, in Service 9, there is testing of each model (Spring 2022 Dataset , 10k new, 20k new) on each of the validation datasets.

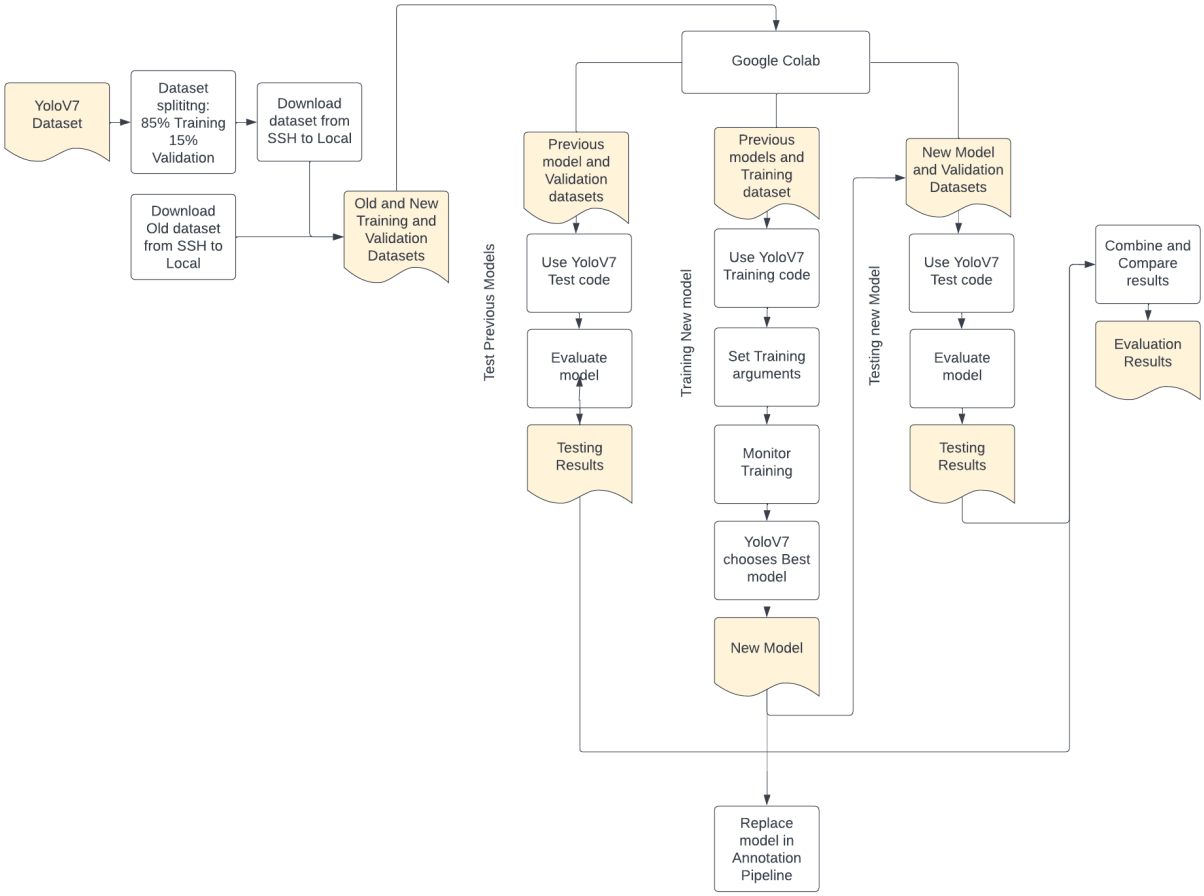

Figure 3: Model Training Pipeline Diagram

4 Implementation

4.1. Splitting PDF repository into equal divisions

The first step in our pipeline is to convert the provided ETD repository into 4 equal pieces. The division of this repository is important because it:

1. prevents the same ETD from being annotated by multiple users.
2. allows file paths to be removed after annotation without touching the original ETD repository.

Steps to split PDF into equal divisions:

1. Create user names for repository division
2. We first grabbed that original ETD repository path by using glob to extract it.
3. Choose a random image from the dataset and assign it to one of the 4 new datasets
4. Pickle the file path array into a *.obj file
5. Specify the number of samples to annotate

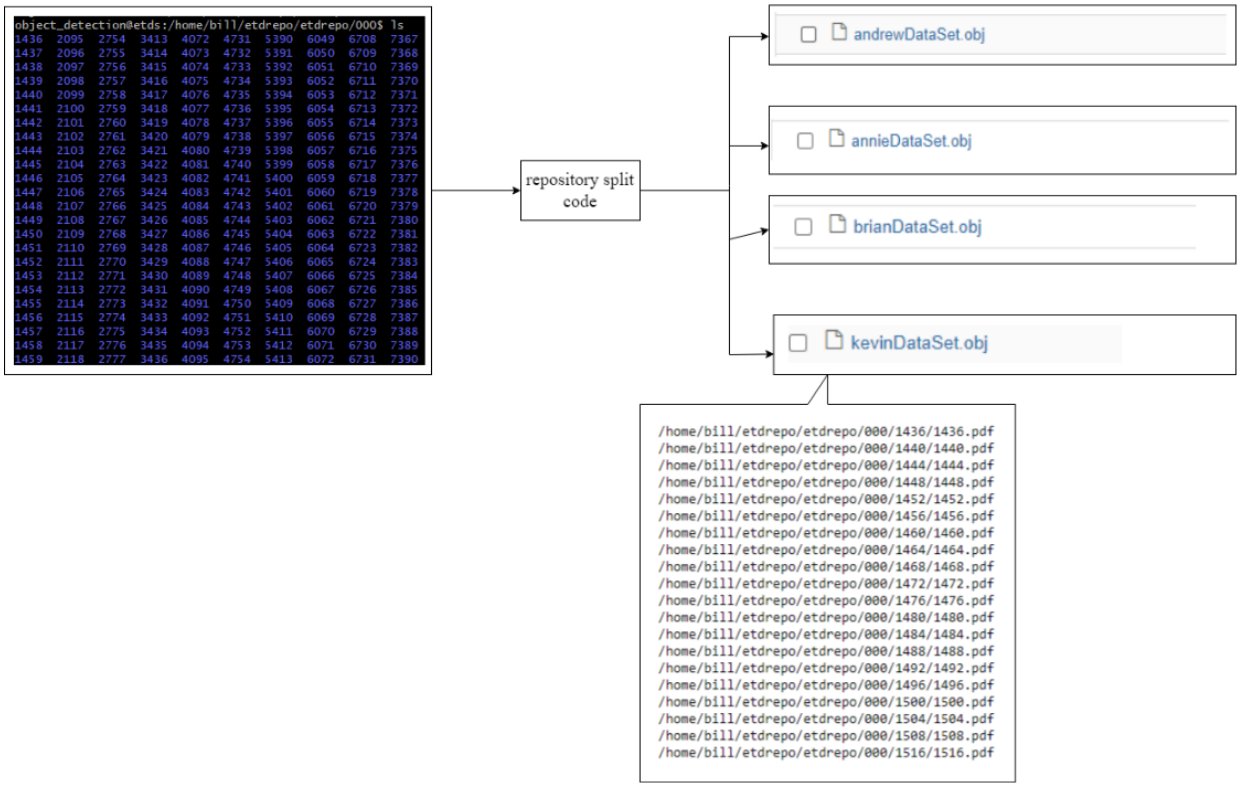

Figure 4: ETD repository to *.obj file Input / Output

4.2 Object Detection Model

Unlike the Object Detection project of Spring 2022, we did not need to use a third-party model and instead used the one that was pre-trained on 200 ETDs. The model name is based on the number of epochs the model had run under. In our case the model name was epoch_149.pt.

```
model_path = '/home/od_5604/yolov7/runs/train/yolov7_merged_coco/weights/epoch_149.pt'
model_conf = 0.25 #@param {type:"slider", min:0, max:1, step:0.01}
model_iou = .45 #@param {type:"slider", min:0, max:1, step:0.01}

model = custom(path_or_model=model_path)

model.conf = model_conf
model.iou = model_iou
```

Once the parameters were set, all the model needed now was to input filtered images for the user to complete AI-aided annotation using the PyLabel library.

4.3 Converting PDFs to Images and Filtering

The next step was to convert PDFs in the folder to a set of image files. This was accomplished using pdf2image. We named the image PDF using the path name of the PDF, for example, if the

pathname of the PDF was /home/bill/etdrepo/etdrepo/000/0001/000002.pdf and there are 132 pages in the PDF, the output of this step will produce a JPEG file named 000_0001_00002_[1-132].jpg.

During the step of converting PDFs to images, PyLabel is also running the model on those images. This produces a set of images with labels, upon which we will then use a basic filtering statement to create a set of images that contain at least 1 of the following minority classes.

Minority classes:

- Title, abstract text, algorithm, author, committee, date, degree, equation, equation number, list of content heading, reference heading, university

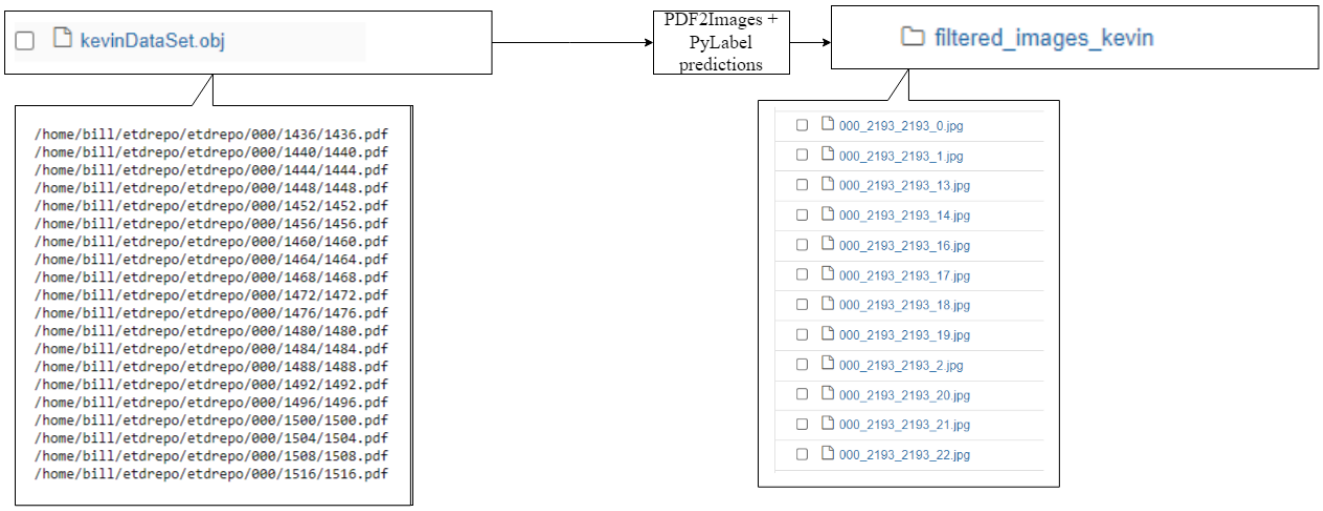

Figure 5: *.obj File to Pages of ETD in .jpg Format

4.4 PyLabel annotation tool

Once the model, YAML file, and filtered images are loaded, we import the filtered images and the model into PyLabel and start the annotation tool using.

```
dataset = importer.ImportImagesOnly(image_dir)
dataset.labeler.StartPyLaber(YOLO_model=model)
```

Once the annotation is loaded, the user is able to use AI-prediction and add, remove, and adjust bounding boxes.

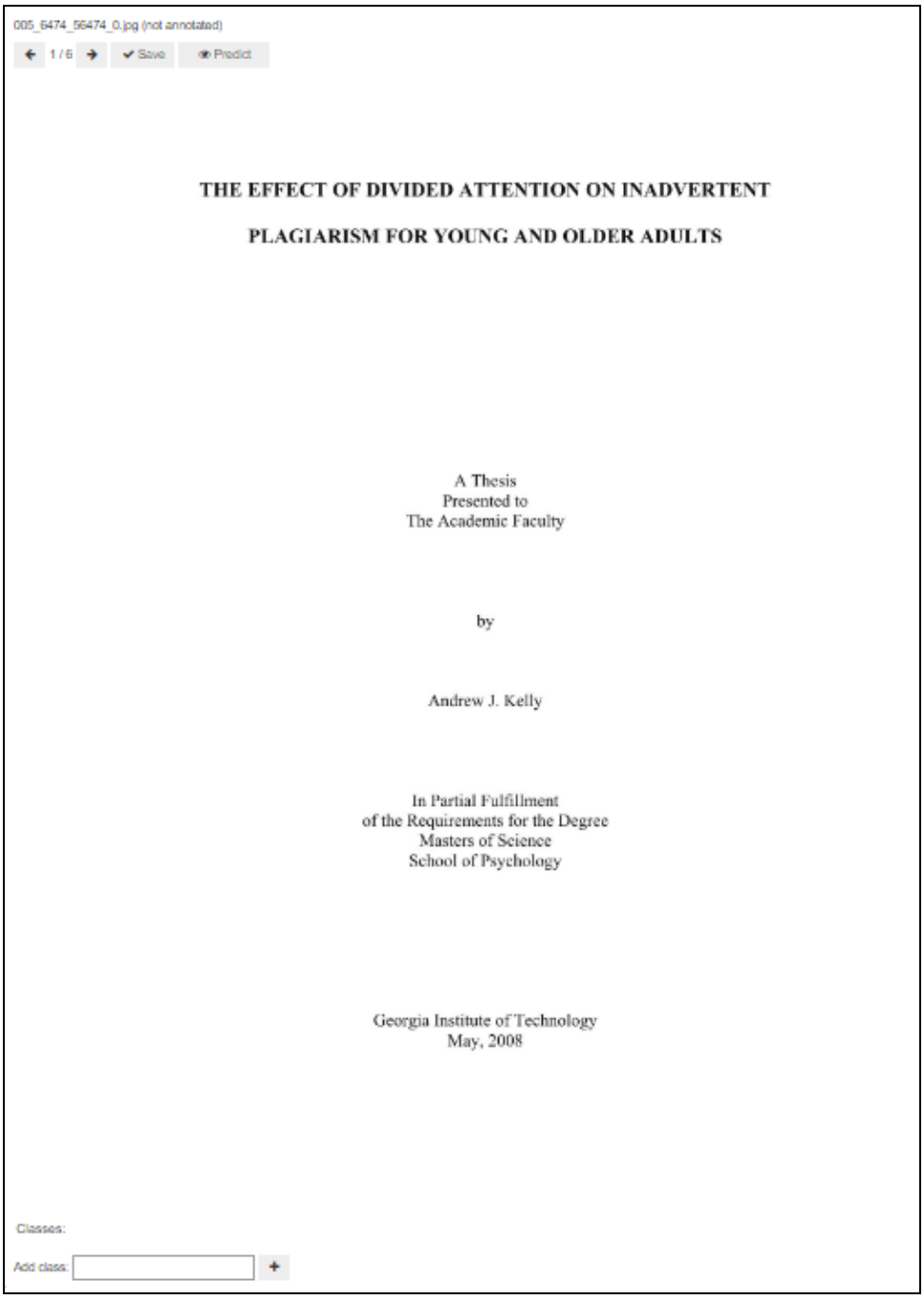

Figure 6: Running PyLabel Pipeline on 1 ETD Sample

In Figure 6, we can see one ETD being run in our pipeline. Observe that there are only 6 pages the model has filtered out that contains minority classes defined in Table 1 out of a total of 78 (not seen) pages of the ETD sample. Also, observe that there are no classes that the user can click to annotate/fix the images.

In the next step, to annotate images, we press the predict button which runs the model and labels the image.

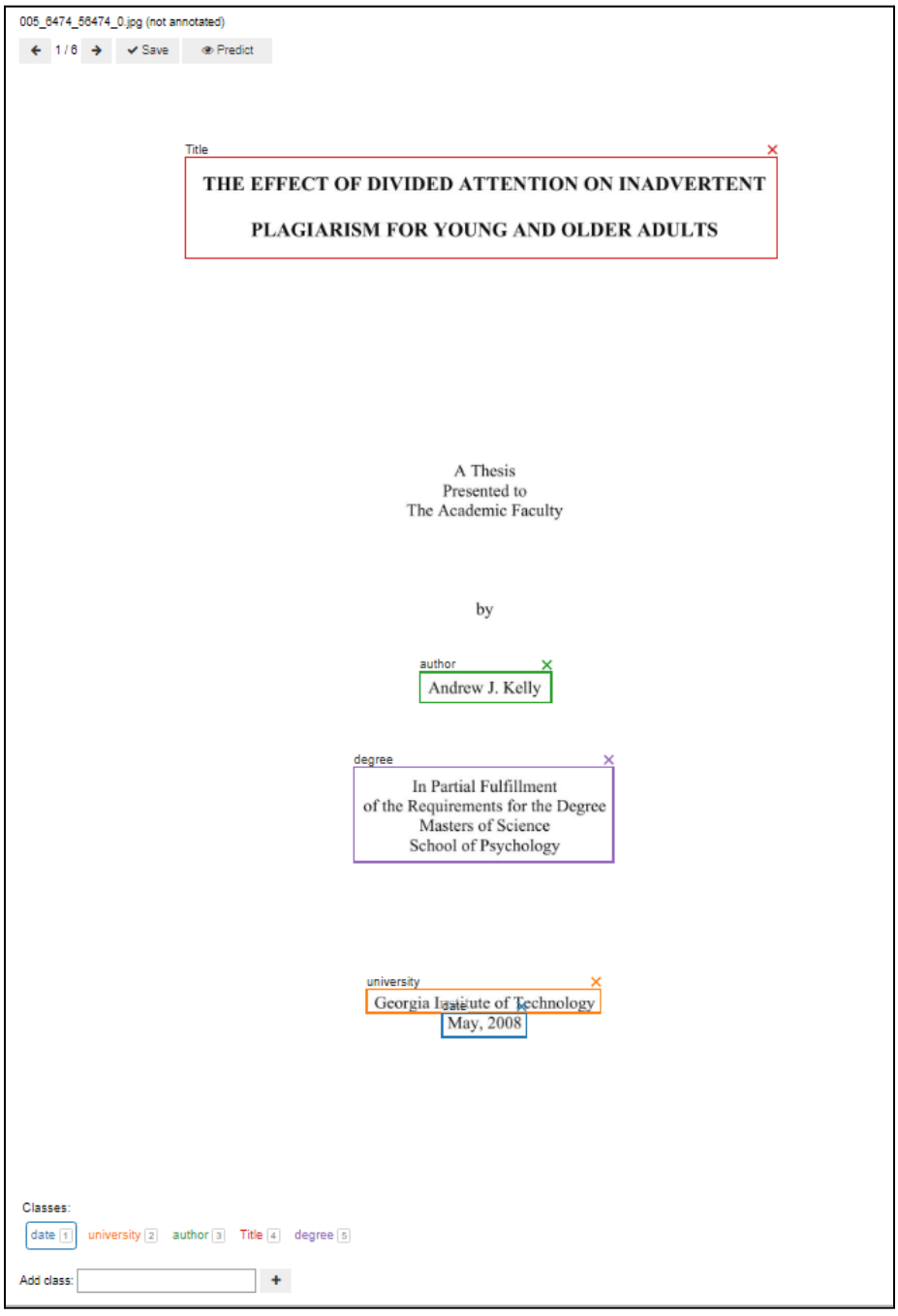

Figure 7: Title Page of ETD After Pressing the Predict Button

Observing Figure 7, we can see the different classes that the title page of an ETD might have. This image is annotated correctly by the model and does not need the user to readjust or manually annotate the image. Also, note that the new labels added to the image also appear at the bottom of the screen combined with the previously used labels. Users may also input classes in a

very specific way in order to add more classes to manually annotate with. More information on this will be in Section 6: Developer's manual.

4.5 Validation of Images and Export to YOLOv7

Once the annotations are finished, we confirm the classes used in annotation are correctly labeled in the YAML file by checking that the string values match. After all the images have been annotated (this is important because un-annotated pages will be counted as a corrupt image by YOLOv7), we proceed to export it to a YOLOv7 training folder; the images will be exported to the images folder and the corresponding annotations will be saved as a .txt file under the same name as the .jpeg file.

5 Testing/Evaluation/Assessment

5.1 Evaluation Discussion

To evaluate the prediction results of the model, the YOLOv7 model tester is used to test model performance. The following two metrics are used to evaluate the performance of the models in YOLOv7.

1. mAP@.5 – mean Average Precision at IoU = .50
2. mAP@.5:.95 – mean Average Precision over different IoU thresholds [.5 to .95]
    a. IoU = {0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95}

Interaction over union (IoU) is used in object detection to indicate the overlapping of the predicted bounding box with the ground truth bounding box defined in the validation dataset. For mAP@.5 where IoU = .50, the evaluator will calculate how many bounding boxes have at least 50% overlap. [6]

$$\textbf{IoU} = \frac{\textbf{Area of Overlap}}{\textbf{Area of Union}}$$

Average Precision (AP) is calculated by generating a prediction score using model, converting those prediction scores to class labels, calculating a confusion matrix where true positives, false positives, true negatives, and false negatives are taken into account, calculating precision and recall, and calculating the area under the precision-recall curve. Precision measures how well a model can find true positives out of all positive predictions. Recall measures how well a model finds true positives out of all the predictions. [6]

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

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

Mean Average Precision (mAP) is then calculated by finding the AP for each class and averaging those numbers together. mAP takes into account the tradeoff between precision and recall, and false positives and false negatives. For this reason mAP is a good metric to evaluate model performance. [6]

$$\textbf{mAP} = \frac{1}{\textbf{N}} \sum_{i=1}^{\textbf{N}} \textbf{AP}_i$$

where

N= number of classes

AP_i = the AP of class i

5.2 Evaluation of Models

5.2.1 Evaluation of Models on the Previous Semester's Validation Dataset

Test names	Old model - Old Validation		10k model - Old Validation		20k Model - Old Validation	
Class	mAP@0.5	mAP@.5:95	mAP@0.5	mAP@.5:95	mAP@0.5	mAP@.5:95
all	0.853	0.526	0.824	0.498	0.803	0.497
Chapter Title	0.888	0.458	0.754	0.368	0.772	0.396
Chapter subheading	0.909	0.423	0.869	0.387	0.809	0.380
Title	0.925	0.491	0.833	0.440	0.841	0.468
abstract heading	0.942	0.362	0.954	0.375	0.942	0.406
abstract text	0.867	0.784	0.879	0.806	0.902	0.826
algorithm	0.666	0.498	0.808	0.634	0.649	0.529
author	0.895	0.385	0.900	0.396	0.936	0.400
committee	0.965	0.592	0.966	0.559	0.975	0.614
date	0.682	0.275	0.695	0.271	0.646	0.280
degree	0.739	0.446	0.755	0.417	0.769	0.453
equation	0.726	0.386	0.773	0.408	0.723	0.398
equation number	0.551	0.198	0.536	0.202	0.475	0.177
figure	0.984	0.810	0.976	0.779	0.945	0.752
figure caption	0.954	0.600	0.929	0.522	0.901	0.538
foot note	0.989	0.769	0.989	0.740	0.976	0.720
list of content heading	0.755	0.310	0.618	0.254	0.703	0.306
list of content heading text	0.993	0.879	0.990	0.874	0.959	0.834
page number	0.513	0.179	0.464	0.150	0.432	0.134
paragraph	0.974	0.821	0.941	0.760	0.924	0.753
reference heading	0.808	0.303	0.782	0.296	0.773	0.319
reference text	0.993	0.913	0.993	0.908	0.994	0.909
table	0.947	0.756	0.886	0.656	0.796	0.580
table caption	0.897	0.488	0.552	0.268	0.469	0.236
university	0.911	0.501	0.937	0.474	0.950	0.525

Table 5: Evaluation Results For Old, 10k, and 20k Model on The Old Validation Dataset

In Table 5 we have the evaluation results for the Old, 10k, and 20k models tested using the old validation dataset. This dataset only includes the samples that the past Spring 2022 semester worked on. Looking at the results we can observe that there are some improvements in the 10k and 20k models, however, there are many results that have actually gotten worse. This may be due to annotator level bias where the results for the old model work better for the old validation

dataset, while our 10k and 20k models perform worse due our team focusing on the minority classes while the other team focused on annotating whole ETDs. Differences in annotation standards may also cause some evaluations to perform worse. As we can see from our 10k and 20k results, the class “table” was significantly worse than the old model’s performance; again this may be because our team was focusing on the minority classes which may make other classes perform the same or worse.

5.2.2 Evaluation of Models on the 10k Validation Dataset

Test names	Old Model - 10k Validation		10k model - 10k Validation		20k Model - 10k Validation	
Class	mAP@0.5	mAP@.5:95	mAP@0.5	mAP@.5:95	mAP@0.5	mAP@.5:95
all	0.892	0.799	0.950	0.782	0.980	0.863
Chapter Title	0.702	0.640	0.858	0.782	0.981	0.843
Chapter subheading	0.957	0.885	0.974	0.825	0.992	0.919
Title	0.968	0.902	0.981	0.798	0.985	0.899
abstract heading	0.977	0.891	0.980	0.889	0.989	0.905
abstract text	0.993	0.992	0.992	0.986	0.996	0.994
algorithm	0.914	0.914	0.995	0.963	0.995	0.987
author	0.889	0.736	0.931	0.678	0.979	0.777
committee	0.878	0.783	0.929	0.699	0.981	0.849
date	0.757	0.567	0.915	0.604	0.936	0.689
degree	0.732	0.598	0.806	0.550	0.871	0.684
equation	0.969	0.877	0.986	0.791	0.994	0.899
equation number	0.954	0.761	0.994	0.807	0.996	0.812
figure	0.880	0.822	0.974	0.917	0.996	0.955
figure caption	0.932	0.874	0.971	0.798	0.996	0.910
foot note	0.917	0.790	0.977	0.722	0.990	0.846
list of content heading	0.979	0.895	0.988	0.801	0.997	0.896
list of content heading text	0.979	0.953	0.990	0.962	0.997	0.971
page number	0.774	0.492	0.916	0.526	0.923	0.554
paragraph	0.979	0.923	0.992	0.889	0.996	0.942
reference heading	0.969	0.843	0.979	0.803	0.996	0.884
reference text	0.961	0.948	0.986	0.973	0.996	0.982
table	0.777	0.739	0.941	0.806	0.996	0.946
table caption	0.787	0.693	0.874	0.668	0.980	0.818
university	0.791	0.646	0.866	0.602	0.961	0.762

Table 6: Evaluation Results For Old, 10k, and 20k Model on The 10k Validation Dataset

In Table 6 we see the results from the evaluation of the Old, 10k, and 20k models tested using the 10k validation dataset. We can see improvements in the 10k, and significant improvements in the 20k model. The 10k and 20k models performed extremely well against the 10k validation dataset. Many of the classes are above 90% in the mAP@50 which fulfills our goals of reaching 90% mAP@50 on the minority classes.

5.2.3 Evaluation of Models on the 20k Validation Dataset

Test names	Old model - 20k Validation		10k model - 20k Validation		20k model - 20k Validation	
Class	mAP@0.5	mAP@.5:95	mAP@0.5	mAP@.5:95	mAP@0.5	mAP@.5:95
all	0.735	0.642	0.809	0.669	0.900	0.763
Chapter Title	0.676	0.535	0.774	0.587	0.926	0.743
Chapter subheading	0.878	0.767	0.910	0.768	0.950	0.840
Title	0.899	0.799	0.935	0.785	0.946	0.837
abstract heading	0.939	0.775	0.952	0.802	0.966	0.826
abstract text	0.824	0.815	0.829	0.811	0.929	0.893
algorithm	0.332	0.312	0.416	0.391	0.699	0.554
author	0.843	0.668	0.902	0.681	0.940	0.736
committee	0.819	0.678	0.910	0.708	0.923	0.748
date	0.717	0.536	0.845	0.597	0.911	0.648
degree	0.541	0.430	0.642	0.481	0.787	0.619
equation	0.898	0.768	0.936	0.747	0.964	0.833
equation number	0.918	0.720	0.976	0.781	0.987	0.796
figure	0.633	0.571	0.694	0.642	0.887	0.797
figure caption	0.810	0.734	0.835	0.696	0.873	0.765
foot note	0.358	0.301	0.430	0.327	0.801	0.667
list of content heading	0.927	0.824	0.949	0.796	0.963	0.866
list of content heading text	0.946	0.922	0.971	0.951	0.966	0.938
page number	0.744	0.470	0.868	0.529	0.900	0.563
paragraph	0.939	0.861	0.960	0.860	0.983	0.905
reference heading	0.919	0.794	0.940	0.783	0.961	0.836
reference text	0.956	0.944	0.961	0.950	0.969	0.951
table	0.704	0.623	0.790	0.720	0.899	0.799
table caption	0.846	0.577	0.945	0.708	0.979	0.763
university	0.755	0.611	0.855	0.630	0.886	0.704

Table 7: Evaluation Results For Old, 10k, and 20k Model on The 20k Validation Dataset

In Table 7 we see the results from the evaluation of the Old, 10k, and 20k models tested using the 20k validation dataset. Although the performance on the 20k testing performed worse compared to the performance on the 10k validation dataset, these results are acceptable because all the models performed worse in general in this test, and our 10k and 20k models performed really well for the minority classes.

5.2.4 Evaluation of Models on the Old + 20k Validation Dataset

After observing the results from Table 5, and discussing with our client Aman, we came to the conclusion that we should merge the Old and 20k validation datasets to normalize the bias in the evaluations.

Test names	Old model - Old+20k Val		10k model - Old+20k Val		20k model - Old+20k Val	
Class	mAP@0.5	mAP@.5:95	mAP@0.5	mAP@.5:95	mAP@0.5	mAP@.5:95
all	0.829	0.630	0.842	0.624	0.869	0.660
Chapter Title	0.806	0.467	0.753	0.426	0.835	0.512
Chapter subheading	0.892	0.531	0.881	0.504	0.866	0.533
Title	0.903	0.730	0.920	0.715	0.927	0.752
abstract heading	0.938	0.671	0.949	0.701	0.958	0.709
abstract text	0.831	0.808	0.837	0.811	0.906	0.868
algorithm	0.380	0.337	0.470	0.421	0.646	0.522
author	0.851	0.591	0.901	0.610	0.939	0.648
committee	0.852	0.645	0.923	0.664	0.936	0.709
date	0.704	0.479	0.821	0.536	0.862	0.566
degree	0.567	0.418	0.654	0.456	0.780	0.578
equation	0.869	0.692	0.908	0.682	0.923	0.747
equation number	0.882	0.659	0.932	0.707	0.933	0.715
figure	0.891	0.744	0.897	0.733	0.930	0.754
figure caption	0.921	0.615	0.907	0.551	0.894	0.575
foot note	0.753	0.588	0.777	0.580	0.905	0.689
list of content heading	0.897	0.710	0.889	0.676	0.914	0.731
list of content heading text	0.961	0.907	0.975	0.927	0.961	0.898
page number	0.594	0.265	0.603	0.260	0.591	0.251
paragraph	0.957	0.835	0.950	0.810	0.956	0.829
reference heading	0.901	0.701	0.919	0.690	0.926	0.726
reference text	0.977	0.924	0.981	0.926	0.985	0.926
table	0.889	0.721	0.853	0.660	0.814	0.622
table caption	0.887	0.499	0.631	0.346	0.575	0.331
university	0.781	0.578	0.870	0.594	0.896	0.660

Table 8: Evaluation Results For Old, 10k, and 20k Model on The Old + 20k Validation Dataset

In Table 8, we can see the results for the Old, 10k, and 20k models tested using the Old + 20k validation dataset. This dataset was created by merging the old validation dataset used in Table 5 and the 20k validation dataset used in Table 7. As expected, the result from testing against the merged validation dataset is between the results from Table 5 and Table 7. This dataset is the best for testing against because it uses both the old and new validation dataset. This is optimal because our model was trained using the old model as a checkpoint.

5.3 Manual annotation vs. AI-aided annotation

To help analyze the improvements of the model we conducted an experiment to test how much faster it was to label images using the AI-aided predictions compared to labeling the annotations manually. For this experiment each of us was tasked to time ourselves annotating pages containing minority classes, with the prediction and without the prediction.

With Prediction (New Model)

Name	# of Pages	Time (h:mm:ss)	Avg Per Page	Avg Per Page (Seconds)
Kevin	285	0:29:46	00:00:06	6
Brian	705	0:59:23	00:00:05	5
Annie	329	0:45:23	00:00:08	8
Andrew	519	0:55:18	00:00:06	6

Figure 8: Screenshot of Data Recorded From New Model Experiment

After we trained the first model with 10,000 new images, we did another round timing ourselves with the new model to see the improvement of predictions. As shown in Figure 8, we recorded the total time along with the number of pages to measure how long it would take to annotate a page on average.

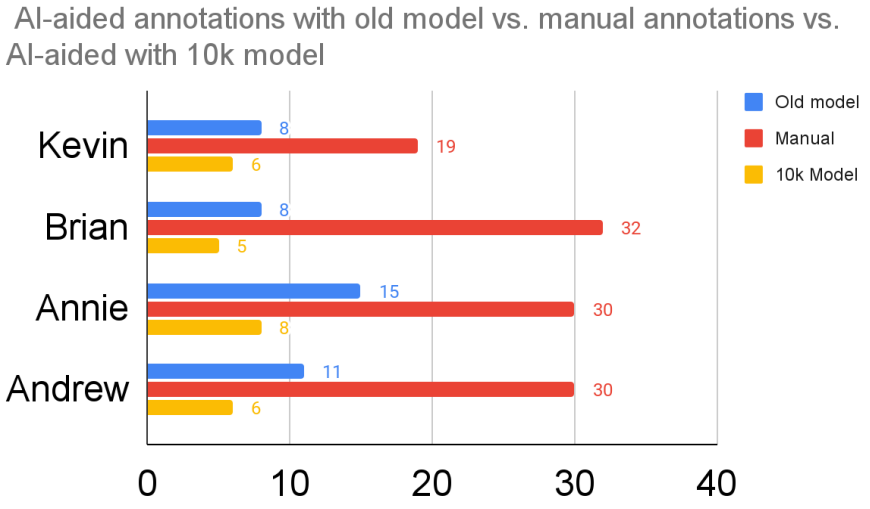

Figure 9: Bar Graph of Average Time Per Page When Annotating

As shown in Figure 9, comparing the average time per page between the AI-aided predictions using the original model and doing the annotations manually, it was around 2-3 times faster using the model. This was likely because the pages that required several labels would be annotated instantly compared to drawing all the bounding boxes manually. Similarly, when using the newly trained model there was around a 2-3 second decrease in time per page. This is likely due to the model getting better at recognizing patterns of commonly grouped classes being paired together. For example, most of the time a figure has a figure caption above or below the figure. The previous model would recognize that there is text and assume it was a paragraph. The newer model would recognize that there was a figure near it, and therefore it is a figure caption rather than a paragraph. This would save time in annotating because it would correctly predict the figure caption rather than a paragraph.

Figure 10: PyLabel Toolbar Example

There were some controls in the experiment that were unaccounted for, but had an impact on the times, such as the toolbar at the bottom of PyLabel which had hotkeys using digits 0-9 on the keyboard to select which bounding box was currently attached to the mouse. An example of this is shown in Figure 10. This affected times because if there are more than 10 classes to select from, some classes may not have hotkeys assigned to them. When annotating certain classes such as paragraph, equation, equation number, etc., that would show much more frequently than others, if they did not have a hotkey assigned, the user would have to scroll to the bottom to manually select the box. This would significantly increase the time and is another reason why having the model produce correct predictions is important to decreasing the time it takes to annotate.

6 Developer's Manual

6.1 Dataset Access

The dataset can be accessed on the object detection server using a Jupyter notebook. However, if the developer is not connected to Virginia Tech's wifi, they must download and set up a VPN in order to use the Jupyter notebook. Once connected to the wifi via VPN, the developer will need to open their Command Prompt (cmd) and input what is written in Figure 11. Figure 11 display what the developer needs to input into in order to set up the server to annotate. Additionally, there are times when the server will need to be reset so the developer must enter the commands written in Figure 11 to reset the server.

```
> ssh object_detection@etds.cs.vt.edu

> screen

> ctrl ad #this is to exit the screen

> screen -ls

> screen -r <specified screen name>

> virtualenv venv --python=python 3.8

> source venv/vin/activate

> pip install jupyter

> jupyter-notebook --no-browser --port 8891 #copy the url that is
  outputted

> ctlr ad #this is to exit the screen

> exit
```

Figure 11: Command Prompt input to set up Object Detection server

Once the developer sets up their command prompt, they will be able to log into the server to access the dataset, build the pipelines on the Jupyter notebook, and perform the annotations on the Jupyter notebook as well. To connect to the server, follow the command written in Figure 12, which displays the command needed to ssh into the Object Detection server .

```
> ssh -L 8891:localhost:8891 object_detection@etds.cs.vt.edu
```

Figure 12: Command Prompt input to ssh into Object Detection Server

When prompted, enter in the password to log in. The password will be provided by one of the server hosts. After successfully logging in, the Jupyter notebook can be opened through a web browser with the outputted URL that was provided in the previous command prompt set up.

6.2 Annotating Data

After accessing the datasets on the object detection server, the next step is to start annotating the data. This is all done within a Python script in the directory ~**/test/** named **pdf_sampling.ipynb** on the server which can be run and edited through the Jupyter Notebook. Table 9 lists the different variables which can be edited to change the settings of the annotation process.

Variables	Description
`sample_count`	Integer value that determines the number of ETDs to be sampled for the annotation process.
`user_name`	String value that determines which .obj pickle file the ETDs will be sampled from. This was used to split the data to make sure we did not annotate the same pages.
`object_categories`	Array that lists the classes which a page must contain at least one of to be selected for annotation. This was used to only annotate pages containing minority classes.
`image_dir`	String value of the path where the selected pages of the ETD will be saved to be used for annotating.
`model_path`	String value of the path to the AI-aided annotation model.
`model_yaml`	String value of the path to the *.yaml file.

Table 9: Variables for Modifying the Annotation Process

From here, running the first 7 cells of code will set the paths, remove any existing images, populate with new images based on username and sample count and then display the first page annotation tool shown. Annotations are done by clicking on the button labeled “Predict” to use the model to predict the bounding boxes. Clicking save will save the bounding boxes of that page in YOLOv7 format. When finished running the next 4 cells will display the dataset, make sure every page is annotated, map the category IDs of the classes to the .yaml file, export the dataset as YOLOv7 to the output path, and output the paths to the newly exported YOLOv7 files.

6.3 Model Training and Evaluation

6.3.1 Data preparation

To split the datasets the Python library split-folders will be used. This is done within a Jupyter notebook named split_dataset.ipynb that has access to the YOLOv7 dataset where all annotations are saved. Figure 13 shows an example of how split-folders can be used to split a dataset into a testing and val dataset.

```
!pip install split-folders

import splitfolders
import random

input = 'training'
out = 'training_split'
seeds= 1337
splitfolders.ratio(input ,output=out,seed=seeds,ratio=(.85,0.15))
```

Figure 13: Example Code of How Split-folders Can Be Used

The variables listed in Table 10 should be edited to change the settings of the split process.

Variables	Description
`input`	A string value of the file path of the input folder
`out`	A string value of the path to the output folder
`seed`	An integer value for shuffling the items. The default value for this is 1337, but should be changed if you want different results.
`ratio`	A ratio of the files you want. In the example above we used 2 numbers for the ratio for Train and Val; however, you can change the ratio to include a third percentage for test if necessary.

Table 10: Variables for the Function splitfolders.ratio()

Once the file split script is done running, a folder at the output path is created with the dataset split and ready for training. From here the folder should be downloaded to a local disk using scp.

Google Colab is hosted on Google's cloud server, so direct access to files on your local disk is not possible. Thus the dataset has to be uploaded to the cloud through Google Drive.

6.3.2 Preparing the training environment

Model training will be done on Google Colab because runtime can be changed to use Google's GPU for model training. This is important because model training requires large amounts of computation power which the Object_Detection server does not have. To prepare your environment for model training you must mount your dataset in your Google Drive to Google Colab using the following lines shown in Figure 14.

```
from google.colab import drive
drive.mount('/content/drive', force_remount = True)
```

Figure 14: Mounting the Dataset to the Google Drive

6.3.3 Installing YOLOv7

To train and test our dataset we will be using YOLOv7. To install YOLOv7 onto your Google Drive, clone the YOLOv7 repository and install the requirements in Google Colab with the following lines in Figure 15.

```
!git clone https://github.com/WongKinYiu/YOLOv7
%cd YOLOv7
!pip install -r requirements.txt
```

Figure 15: YOLOv7 Installation Commands

This will install YOLOv7 and its requirements onto your Google Drive. It is recommended that you place your *.yaml that contains the classes and file paths to train and val into the YOLOv7 directory. This .yaml file should be edited if you are using multiple datasets for training and testing.

6.3.4 Model Testing

YOLOv7 includes a testing script called test.py. To run this script the following flags in Table 11 should be defined.

Flags	Description
`--data`	The file path to your *.yaml file. This .yaml file should be edited if you want to change what dataset you are testing on.
`--img`	The image size of your dataset. Default is 640.
`--batch`	An integer value for batch size used in training.
`--weight`	The file path to your training weight.
`--name`	The name of the output file where the results are saved.

Table 11: Flags for test.py

After these flags have been defined the test script can be run. Make sure the runtime type is set to GPU in order to make use of Google Colab's GPUs. Figure 16 shows an example of running the testing script:

```
!python test.py --data data/dataset.yaml  --img 640 --batch 32
--weights Training_weights/20k_weight.pt --name test_20k_weight
```

Figure 16: Python Line to Run YOLOv7 Testing Script

6.3.4 Model Training

YOLOv7 includes a training script called test.py. To run this script the following flags in Table 12 should be defined.

Flags	Description
`--data`	The file path to your *.yaml file. This .yaml file should be edited if you want to change what dataset you are training on.
`--workers`	An integer value indicating number of CPU workers allocated for training
`--cfg`	The config file for the model architecture. The possible options are yolov7-e6e.yaml, yolov7-d6.yaml, yolov7-e6.yaml, yolov7-w6.yaml, yolov7x.yaml, yolov7.yaml, and yolov7-tiny.yaml. The architecture should be selected to best fit the complexity of the object detection task.
`--hyp`	A yaml file that describes the hyper-parameter for model training. Default file is data/hyp.scratch.yaml.
`--epochs`	Number of epochs to train for. YOLOv7 will distribute the dataset across all the passes the model training will take.
`--img`	The image size of your dataset. Default is 640x640.
`--batch`	An integer value for batch size used in training.
`--weight`	The file path to your pretrained model weight. If the developer wants to start from scratch this flag should be followed with an empty string ‘ ’.
`--name`	The name of the output file where the results are saved.

Table 12: Flags for train.py

After the flags in Table 12 have been defined, the training script can be run. Make sure the runtime type is set to GPU in order to make use of Google Colab’s GPUs. In Figure 17 an example of running the training script can be seen.

```
!python train.py --workers 8  --batch-size 32 --data
data/dataset.yaml --img 640 640 --cfg cfg/training/yolov7.yaml
--weights runs/train/new_model_training4/weights/last.pt --name
20k_model --hyp data/hyp.scratch.custom.yaml --epochs 100
```

Figure 17: Model Training Function

6.3.4 Model Training Resume

A flaw with using the free version of Google Colab is that it requires constant monitoring in order to keep the notebook running. Google Colab will detect when changes are being done to the notebook. After a random amount of time Google Colab will prompt the user to keep the notebook running. This may cause your training script to crash. In order to resume the training from the last checkpoint, the resume flag should be defined with the file path to the checkpoint as shown in Figure 18.

```
!python train.py --resume runs/train/sample_dir/weights/last.pt
```

Figure 18: Resume flag used to resume training from a specific checkpoint

7 Lessons Learned

7.1 Problems / Solutions

7.1.1 PyLabel vs. Roboflow

This semester we switched the annotation process to use PyLabel rather than Roboflow. This made the process of annotating much simpler because the annotations could be done on a Jupyter notebook rather than its own program. The selection of bounding boxes was also easier because of the hotkeys used to select certain bounding boxes which sped up the annotation process.

7.1.2 Model Training

For model training we decided to use Google Colab. Google Colab has paid premium features which allows the training to run in the background without any CAPTCHA popping up and also allows access to premium computer units which allows for faster training. We decided not to use this because of the cost, however this resulted in much slower training times which required constant supervision to make sure the process was not stopped due to CAPTCHA requests. This made our total model training time much longer as well as much more stressful. We could recommend anyone training in the future to invest in the premium features.

7.1.3 Annotation Standard Guide

As discussed in Section 5.2, one of the problems that occurred was when the new models were tested on data annotated by the previous semester's group, the model performed worse, and continued to perform poorly after each iteration. This was due to the fact that there was no standard for annotating and therefore the annotations for pages varied between semesters. To fix this for the next group, we suggest creating an annotation standard for ETDs to make sure that all annotations are following the same guidelines. If the annotation standard for ETDs does not provide an example on how to annotate certain elements, the team will still be able to ask the client for guidance if they are still unsure on how to annotate those elements.

8 Future Plans

8.1 Grouping Commonly Paired Classes

The main idea that we proposed for the subsequent semester is the idea of training the model to recognize groupings of two classes that are commonly found together. While still annotating the minority classes, we also would include a new box to help label commonly paired classes.

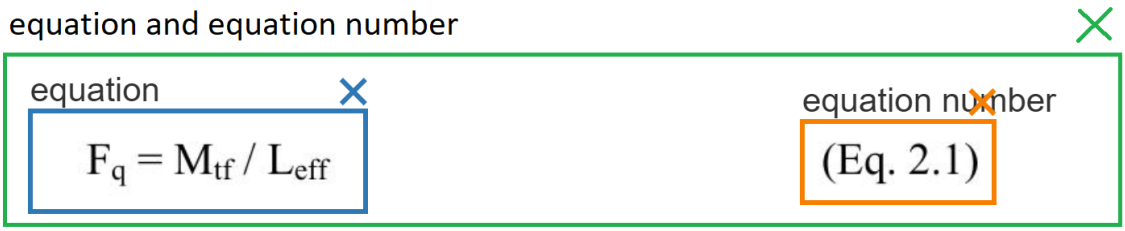

Figure 19: Grouping of Equation and Equation Number

As shown in Figure 19, an example of this would be grouping equations and equation numbers since they are commonly found together. When annotating in the future we would include these boxes on any classes that are grouped together to help train the model to recognize patterns like this while still annotating the minority classes. This would be an idea for the future semester because a majority of the annotations that we made did not include these additions.

Full List of Grouping Pairs:

1. Figure + Figure Caption
2. Table + Table Number
3. Equation + Equation Number
4. Reference Heading + Reference Text
5. List of Content Heading + List of Content Text
6. Abstract Heading + Abstract Text
7. University + Degree

8.2 Final Model Testing

Following our training of the model using the 20,000 images of annotating, we want to conduct a similar experiment to what we conducted in Section 5.4. We want to time ourselves using the new model and see the improvement compared to the old model, manual, and 10,000 model. This would be a good metric to see how fast the final model is after reaching the final goal of total annotations for the semester.

Acknowledgments

Client: Aman Ahuja
Email: aahuja@vt.edu

Professor: Dr. Edward A. Fox
Email: fox@vt.edu

References

[1] Roboflow team, “Give your software the power to see objects in images and video,” Roboflow, 2020. [Online]. Available: https://roboflow.com/. [Accessed: 04-Dec-2022].

[2] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, “Detectron2,” *GitHub*, 2019. [Online]. Available: https://github.com/facebookresearch/detectron2. [Accessed: 01-Dec-2022].

[3] Heaton, A., Fraenkel, J., and Topper, D. PyLabel. https://github.com/pylabel-project/pylabel, 2019. [Accessed: 04-Dec-2022].

[4] Jupyter, Project Jupyter. https://jupyter.org/, 2022. [Accessed: 03-Dec-2022].

[5] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” arXiv preprint arXiv:2207.02696, 2022. [Accessed: 04-Dec-2022].

[6] D. Shah, “Mean average precision (MAP) explained: Everything you need to know,” *V7*, https://www.v7labs.com/blog/mean-average-precision. 07-Oct-2022. [Accessed: 12-Dec-2022].