Visual Understanding of Academic Diagrams using Multimodal Generative Models

Hashem Elezabi **Ahmed Sharaf**
Department of Computer Science
Stanford University
`{hashem, ammas}@stanford.edu`

1 Introduction

The recent advances in generative artificial intelligence (AI) have led to unprecedented AI capabilities in language and vision. Large language models (LLMs) like ChatGPT have revolutionized natural language processing (NLP). At the same time, multimodal vision-language models that can understand both language and images are being rapidly developed [4], with OpenAI GPT4-V and Google Gemini being the most powerful. All these models are examples of transformer-based deep generative models.

This project aims to build a deep generative model that takes an image of an academic diagram and outputs text or source code that can be used to recreate this diagram, such as LaTeX code or API code for creating a presentation slide. An academic diagram can include shapes like circles, rectangles, and arrows, and it can also contain more specific formats like graphs (with nodes and edges) or tables. The practical use case that motivates this project is the ability to feed a handwritten sketch to a model and have it output source code that describes the sketch, so that one can generate a real slide or proper LaTeX just from the input sketch.

We simplified this problem by considering specific types of diagrams. In particular, we considered two different types of diagrams: 1) a slide that includes shapes like rectangles, circles, arrows, etc. and 2) a `tikz` diagram. We created two datasets for the two diagram types, and we trained a different kind of model for each type. The first model (for slides) is trained to detect objects in the image, which we would then map to source code deterministically using a simple mapping. The second model (for `tikz`) is trained to generate a text description of the input graph in a specific format, which is then mapped to `tikz` source code deterministically as well using a simple mapping. We discuss these details in Sections 3 and 4.

Note: Our code for this project is at https://github.com/hashemelezabi/cs236-project.

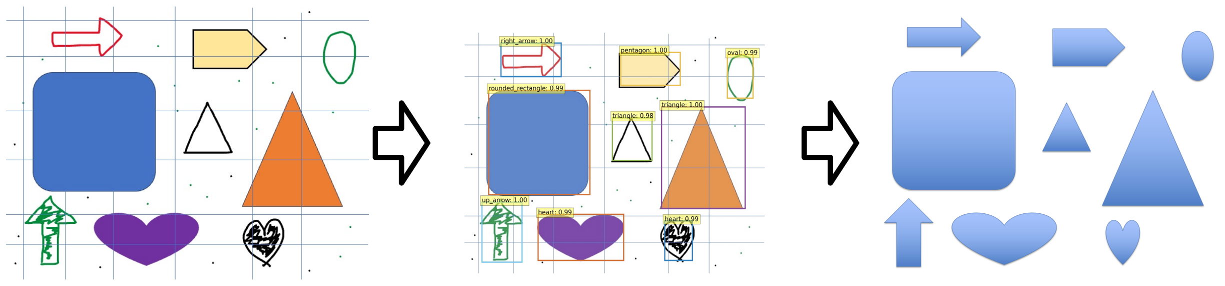

Figure 1: Showing the whole process of converting drawings to a slide.

The whole process depicting input sketch to output constructed slide is shown in Figure 1.

2 Related Work

Visual document understanding. The recently introduced Nougat model by Blecher et al. [1] is a visual Transformer model that performs an Optical Character Recognition (OCR) task for processing scientific documents into a markup language. This model is an encoder-decoder Transformer built on the Donut document understanding architecture [3], which introduced a transformer model for visual document recognition that avoids the use of separate OCR modules. It uses an encoder based on the Swin Transformer [6] that converts a document image into latent embeddings. These embeddings are subsequently converted to a sequence of tokens in an autoregressive manner in the decoder, using self-attention to attend to different parts of the sequence and cross-attention to attend to different parts of the encoder output. The decoder is based on the mBART architecture [5]. While Nougat processes images of scientific documents in PDF format and converts them to a markdown format, we are focusing on detecting objects in slides and academic diagrams.

Object detection. There has been much work on object detection using deep learning-based computer vision techniques. Faster RCNN [7] is a model architecture that became very successsful for object detection, though it is complex to understand and implement. The transformer-based DETR [2] model was recently introduced and offers similar performance to Faster RCNN while being a simpler end-to-end model. We heavily relied on DETR in our project.

Multimodal generative models. Another body of work related to our project is the growing work on multimodal models that combine a vision encoder and language decoder to achieve general-purpose visual and language understanding. These models differ from models like Nougat in that they serve as foundation models for general-purpose vision-language understanding, as opposed to being trained for a specific input and output type. The best such model is GPT-4V, but there are recently introduced open-source models that are approaching the performance level of GPT-4V. The two main examples of this are LLaVA [4] and MiniGPT-4 [9]. We experimented with GPT-4V, LLaVA, and MiniGPT-4 in an inference, where we provide instructions to the model in the prompt and ask it to generate the output in the desired format with no given examples (zero-shot) or a couple examples (few-shot). None of them worked to output correct predictions, indicating that these vision-language foundation models are not suitable for this task without further fine-tuning.

3 Datasets

We created two datasets for the two different diagram types.

3.1 Slides Dataset I

For this dataset, the problem statement is as follows. Given an image of a slide, we want to predict the shapes in the slide and their locations, so that we are able to reproduce the slide in the native slide format. If we have the shapes and locations, we can easily generate code that creates the corresponding slide. For Google Slides, we would create Google Slides API requests, while for PowerPoint, we would use the `python-pptx` package.

We used `python-pptx` to generate 10,000 slides, each of which contains n shapes where n is randomly picked from $1, \ldots, 10$. Each shape can be a pentagon, heart, isosceles triangle, oval, down arrow, left arrow, up arrow, right arrow, rounded rectangle, or rectangle. For this dataset, we are using an object detection model, which cannot readily detect text, so we only generate empty shapes.

Since shapes don't typically overlap in a presentation slide, and since overlapping shapes would be challenging for our model to understand, we ensure there is no overlap by only adding a shape if it doesn't overlap with any existing shapes in the slide, and continuing to generate random locations and sizes until that is achieved.

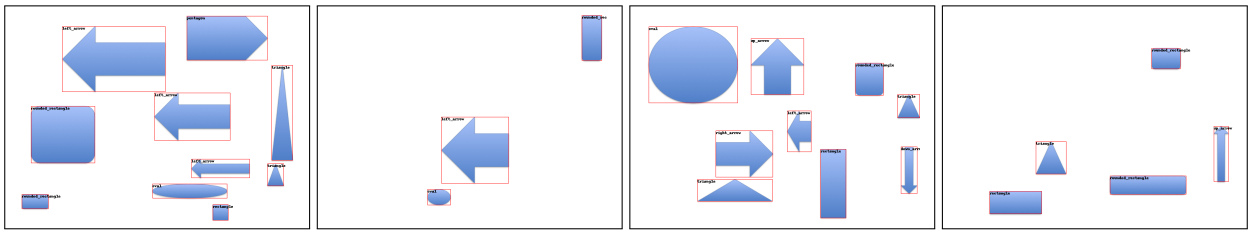

Figure 2: Examples of annotated slides from Dataset I

Figure 2 shows example generated slides and their bounding box annotations. Each slide represents one data point x. The corresponding y, which is visualized as the set of red annotations seen in the figure, is a set of ground-truth object categories and their bounding box descriptors. Each element in this set is an annotation in the form `[shape_index, left, top, width, height]`, where `shape_index` is the index of the shape out of a list of possible shapes (e.g. index of rectangle), `left` is the x value of the top-left corner, `top` is the y value of the top-left corner, and `width` and `height` are the width and height of the bounding box of the shape. As an example, the annotations for the rightmost slide in 2, shown enlarged in 3, are:

```
[10, 84.74, 463.51, 135.32, 59.18]
[3, 205.98, 333.77, 78.41, 84.48]
[9, 399.17, 423.06, 199.36, 48.81]
[7, 671.41, 293.69, 37.98, 144.63]
[9, 508.79, 89.33, 75.17, 54.49]
```

where 10 is the index of *rectangle*, 3 is the index of *triangle*, 9 is the index of *rounded rectangle*, and 7 is the index of *up arrow*. Note that all images have width 720 and height 540.

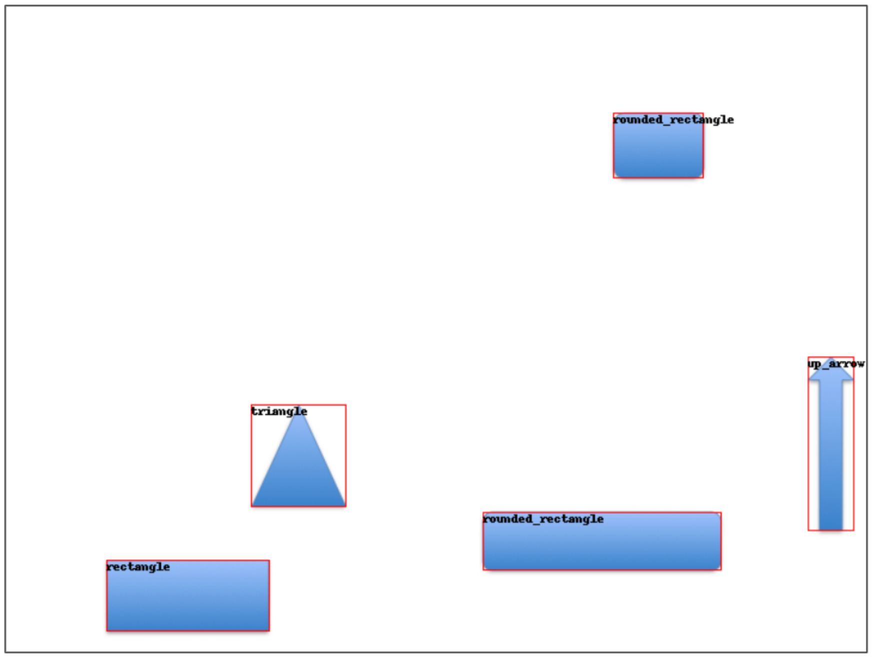

Figure 3: Enlarged example slide with annotations.

3.2 Slides Dataset II

The second dataset was created as an attempt to improve the model's capabilities and make it able to detect the shapes even if different colors or different artistic styles are presented. The second dataset follows the first one in terms of the structure and the process of generation. The same technique was used for generating the shapes, except for this one, there was a 50-50 chance that the shape would be filled with solid color or be left only as an outline. Afterwards, the color of the shape is also randomly generated to make sure that the model does not adhere to any specific colors. Also, the number of shapes in each slide was randomly selected from 1 to 10; then, each shape was randomly placed and relocated until no two shapes overlap. The same python library was used to to generate the final Powerpoint file after 10K slides were generated as mentioned.

One of the bigger reasons for adding some shapes just as outlines is that we want the model to be ready to detect shapes drawn by hand, which usually are just the outlines without fill.

Same as the first dataset, the each slide is created with a list of 5-tuples, defining the shape type, and the bounding box. A sample of this dataset is shown in Figure 4.

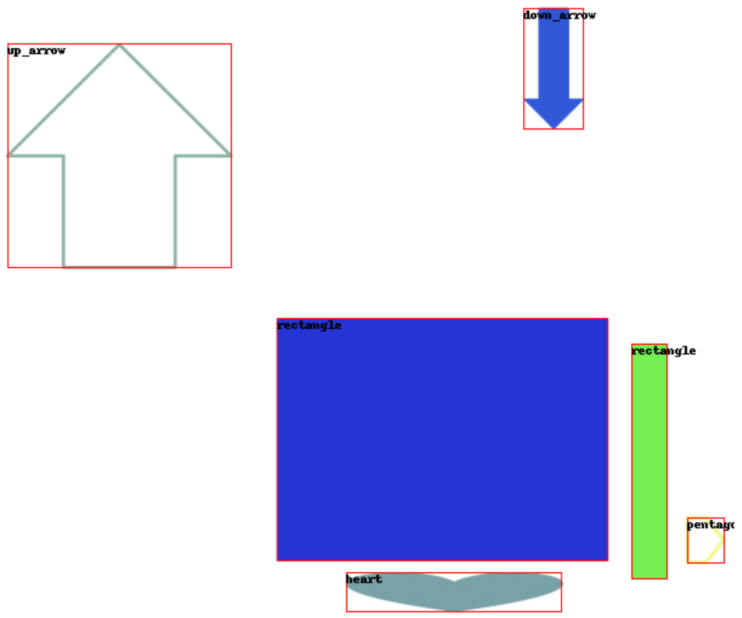

Figure 4: Enlarged example slide with annotations of dataset 2.

3.3 `tikz` Dataset

For the `tikz` dataset, the problem statement is as follows. Given an image of a graph drawn using tikz, output tikz code that can be used to recreate that graph. To simplify the problem, we restrict ourselves to graphs with a specific structure: topologically sorted graphs consisting of layers of nodes where each layer's nodes point to the nodes in the next layer. We randomly choose the total number of nodes to be a number from 1 to 6, and we randomly pick the number of nodes to put in each layer.

Each node is labeled with a label of the form

```
<letter>{<sub_or_sup><digit>}
```

where letter can be one of a, b, c, x, y, z, `sub_or_sup` can be a subscript or superscript, and digit can be from 0 to 9. We created 5000 images of these tikz graphs. Figure 5 shows some examples.

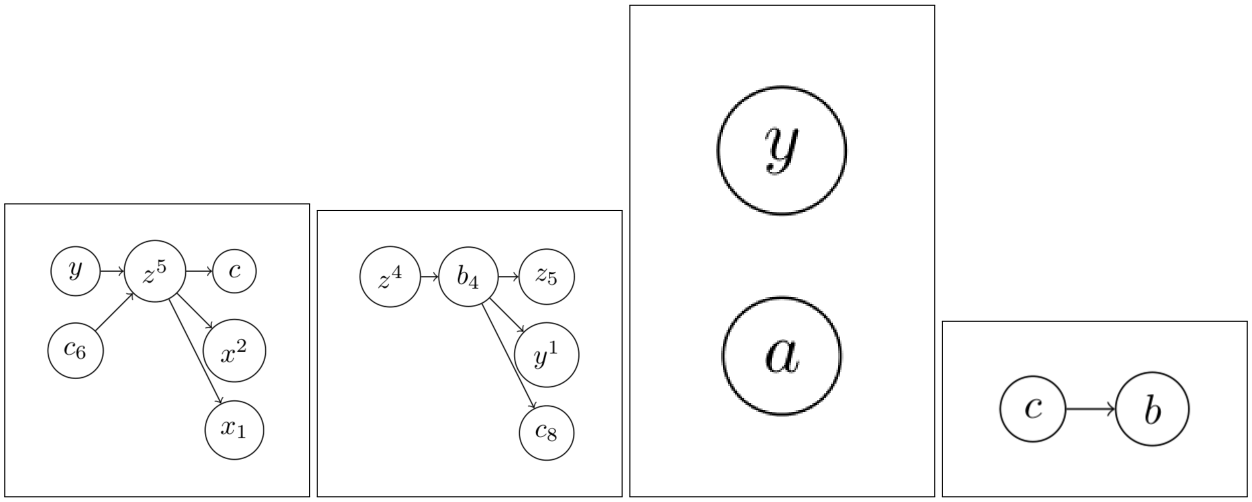

Figure 5: Examples of tikz diagrams from our dataset

Since we limit the graphs to this structure, the output format is also restricted to a specific structure that can be easily mapped to valid tikz code. For the leftmost image above, the ground-truth text description is

```
{ y , c_6 } -> { z^5 } -> { c , x^2 , x_1 }
```

4 Models

4.1 DETR object detection model

Our initial approach is to use a transformer encoder-decoder architecture to output a set of bounding box predictions, as described above. This is an object detection approach to our problem, and we found that we can use the transformer-based DETR model [2] as a baseline. DETR stands for DEtection TRansformer [2]. This model combines a convolutional backbone with an encoder-decoder transformer to build an object detector that is much simpler than the complex Faster R-CNN model while achieving similar performance. We use a pretrained DETR model provided by HuggingFace and fine-tune it on our slides dataset, after the preprocessing it to match what DETR expects and applying random horizontal flipping as data augmentation.

DETR starts by feeding the input image into a pretrained ResNet convolutional backbone to obtain a feature map. This is then projected to match the hidden dimension of the transformer of DETR using another 2D convolution layer. The result is sent to the encoder, which outputs a further set of encoded features. Next, embeddings representing *object queries* are sent through the decoder, which outputs updated hidden states. Two classifier heads are added at the end, a linear layer predicting one of the object categories or "no object", and an MLP to predict bounding boxes for each query. To train the model, DETR uses a *bipartite matching loss* that compares predicted and ground-truth bounding boxes. Specifically, DETR infers a fixed-size set of N predictions, where N is significantly larger than the typical number of objects in an image. The DETR loss produces an optimal bipartite matching between predicted and ground truth objects, and then optimizes object-specific (bounding box) losses.

Let y denote the ground truth set of objects, and let $\{\hat{y}\}_{i=1}^{N}$ denote the set of N predictions. We can consider y as also a set of size N padded with $\varnothing$ representing no object. DETR searches for a permutation of the N predictions with the lowest cost:

$$\hat{\sigma} = \arg\min_{\sigma \in \mathcal{S}_N} \sum_i \mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$$

where $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ is a pairwise matching cost between the ground truth y_i and a prediction with index $\sigma(i)$. This cost takes into account both the class prediction loss and the bounding box loss, and is defined as

$$-\mathbf{1}_{\{c_i \neq \emptyset\}} \hat{p}_{\sigma(i)}(c_i) + \mathbf{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$$

where $\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$ is a linear combination of the ℓ_1 loss and the generalized IoU loss that is scale-invariant [8]. Figure 6 depicts the DETR method.

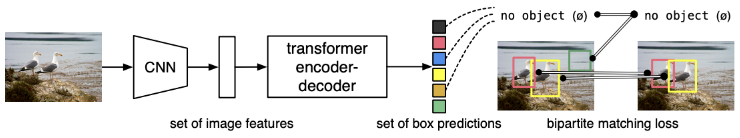

Figure 6: DETR architecture [2]

4.2 Visual encoder and language decoder based on Nougat

In the Nougat model, the encoder is a Swin transformer [6] and the decoder is a series of self-attention and cross-attention layers. The loss function has two parts: cross-entropy loss for the class prediction (i.e. shape type prediction), and L_1 loss for the bounding box prediction.

We build our own model following the Nougat architecture. We use the same Swin encoder and also use the MBart decoder, but we use our own tokenizer where the only possible tokens are the 130 possible node labels as described above.

5 Results and Discussion

To evaluate predictions, we use the common object detection metric of mean average precision (mAP) at an IoU threshold of 0.5. IoU stands for Intersection over Union, and it quantifies how close a predicted bounding box is to the ground-truth bounding box by computing the intersection area and dividing by the union area. Our predicted bounding box is represented by the last four elements of the vector described above. The mAP value is the mean of the average precision (AP) values computed for each class, which is the shape type in our case. The AP value computed at an IoU threshold of 0.5 means that we consider a bounding box a positive prediction if the IoU between it and the ground truth bounding box is over 0.5, and a negative prediction otherwise.

We fine-tuned DETR on 8K slide images from our dataset 1 for 5 epochs, then tested it on the 2K remaining images. We achieved a mean Average Precision value of 75.6 percent (up to 89 percent at IoU=0.5) and a mean Average Recall of 59.5 percent (up to 80.6 percent if given 10 detections).

We then used the second dataset to fine-tune another DETR model on 8K slides, then we achieved a mean Average Precision value of 73.7 percent (up to 89 percent at IoU=0.5) and a mean Average Recall of 58 percent (up to 79 percent if given 10 detections).

That was a huge success because when the older model was evaluated on dataset 2, it only got a percision value of 60.4 percent (up to 76.9 percent at IoU=0.5) and a mean Average Recall of 50.1 percent (up to 68.1 percent if given 10 detections).

	IoU threshold (AP) or maxDets (AR)	Model 1 on dataset 1	Model 1 on dataset 2	Model 2 on dataset 2
AP	0.5:0.95 (average)	75.6	60.4	73.7
AP	0.5	89	76.9	89
Avg Recall	1	59.5	50.1	58
Avg Recall	10	80.6	68.1	89

Table 1: Showing all the results we got for the two models.

Table 1 shows all the values we have got.

Also the Graph 7 and Graph 8 show the training loss for model 1, the one that was trained on dataset 1 are shown here after the model was trained for 5 epochs.

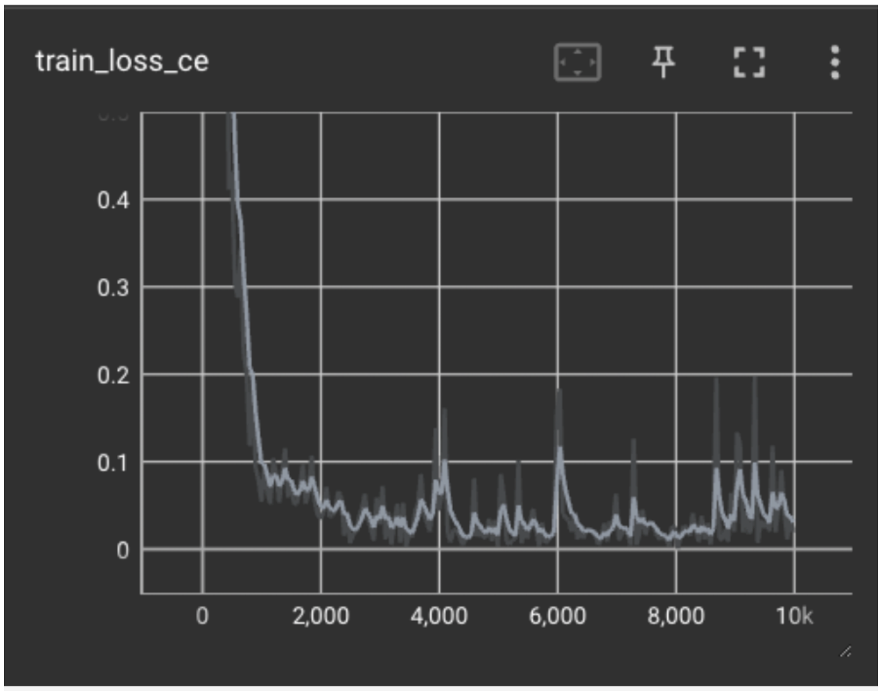

Figure 7: Training cross-entropy loss for model 1

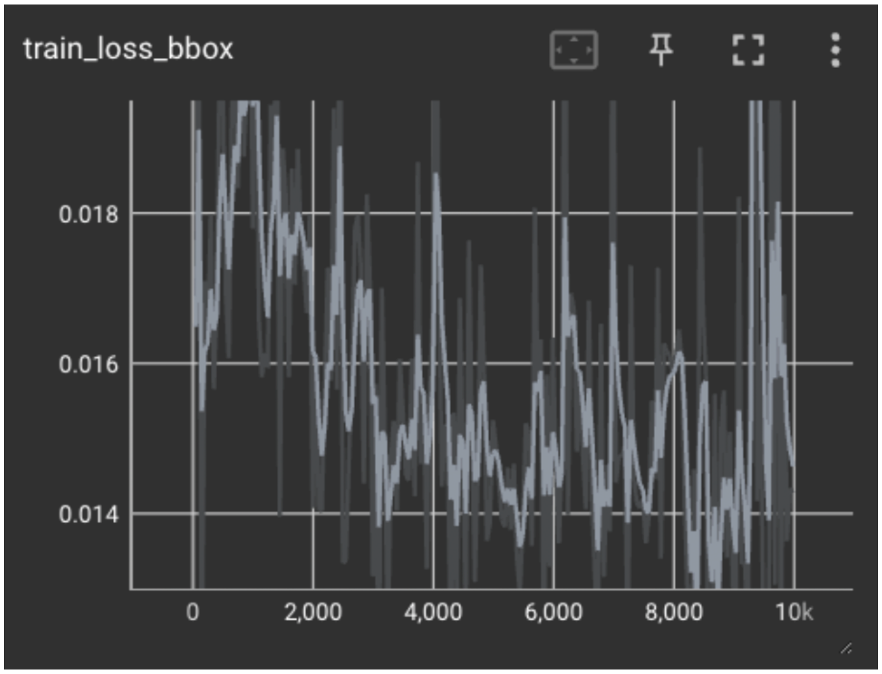

Figure 8: Training bounding box loss for model 1

The Graph 9 and Graph 10 show the training loss for model 2, the one that was trained on dataset 2 are shown here after the model was trained for 5 epochs.

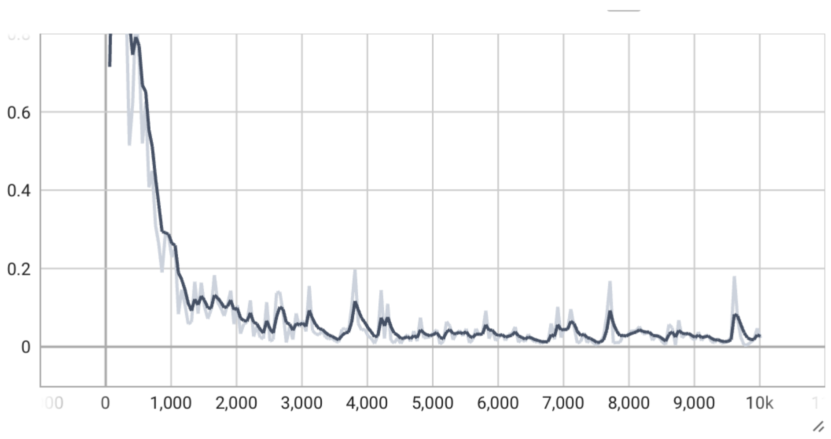

Figure 9: Training cross-entropy loss for model 2

Because dataset 2 had outlines that resembled hand-drawn shapes, we decided to try the model with actual hand-drawn shapes and even hand-drawn shapes that were partly filled. After that, noise consisting of lines and different colored dots scattered everywhere were added to the slide to try and confuse the model; however, the model showed great success again in determining the shapes and converting them to an actual Powerpoint slide as shown in Figure 1.

6 Conclusion

We discussed two approaches for visual document understanding, specifically focusing on academic diagrams. Our DETR method worked very well and showed great results. The Nougat-based model for the tikz dataset is still training, so we're still looking to get results on that.

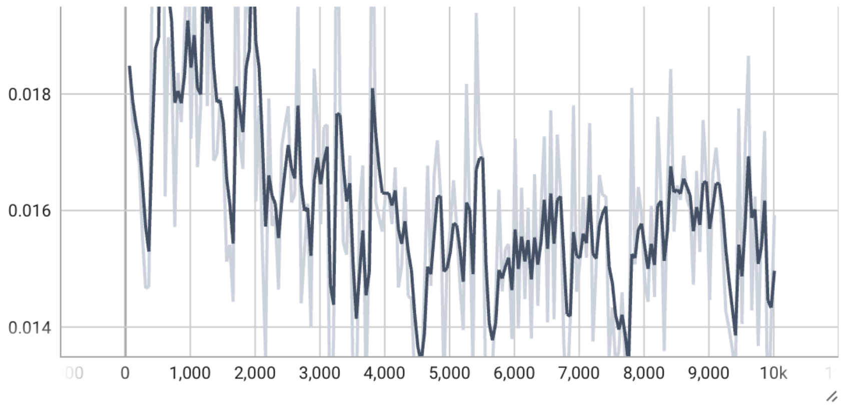

Figure 10: Training bounding box loss for model 2

References

[1] Lukas Blecher et al. *Nougat: Neural Optical Understanding for Academic Documents*. 2023. arXiv: `2308.13418 [cs.LG]`.

[2] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. arXiv: `2005.12872 [cs.CV]`.

[3] Geewook Kim et al. *OCR-free Document Understanding Transformer*. 2022. arXiv: `2111.15664 [cs.LG]`.

[4] Haotian Liu et al. *Visual Instruction Tuning*. 2023. arXiv: `2304.08485 [cs.CV]`.

[5] Yinhan Liu et al. *Multilingual Denoising Pre-training for Neural Machine Translation*. 2020. arXiv: `2001.08210 [cs.CL]`.

[6] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: `2103.14030 [cs.CV]`.

[7] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: `1506.01497 [cs.CV]`.

[8] Hamid Rezatofighi et al. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*. 2019. arXiv: `1902.09630 [cs.CV]`.

[9] Deyao Zhu et al. "MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models". In: *arXiv preprint arXiv:2304.10592* (2023).