Technical Drawing Text Extraction Using Optical Character Recognition

Presenter: Daniel A. Masters, org 9358

Project Mentor: April Suknot, org 9354

Sandia National Laboratories

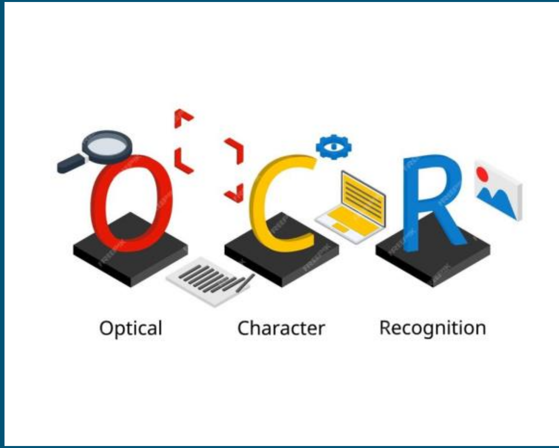
Presenter Contact: damaste@sandia.gov
danielangmasters@gmail.com

Date: 10/29/2024

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

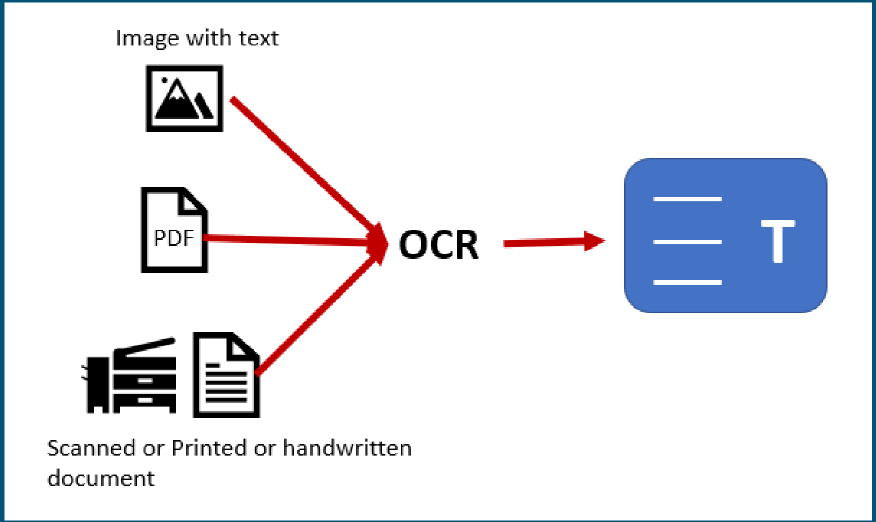
Overview

1) What is Optical Character Recognition (OCR)?
2) Context of OCR and its Use-Case for Sandia National Laboratories
3) Dataset Creation and Preprocessing
4) Model Selections
5) Model Evaluations
6) Results, Future Implementation: Version 2
7) What I learned



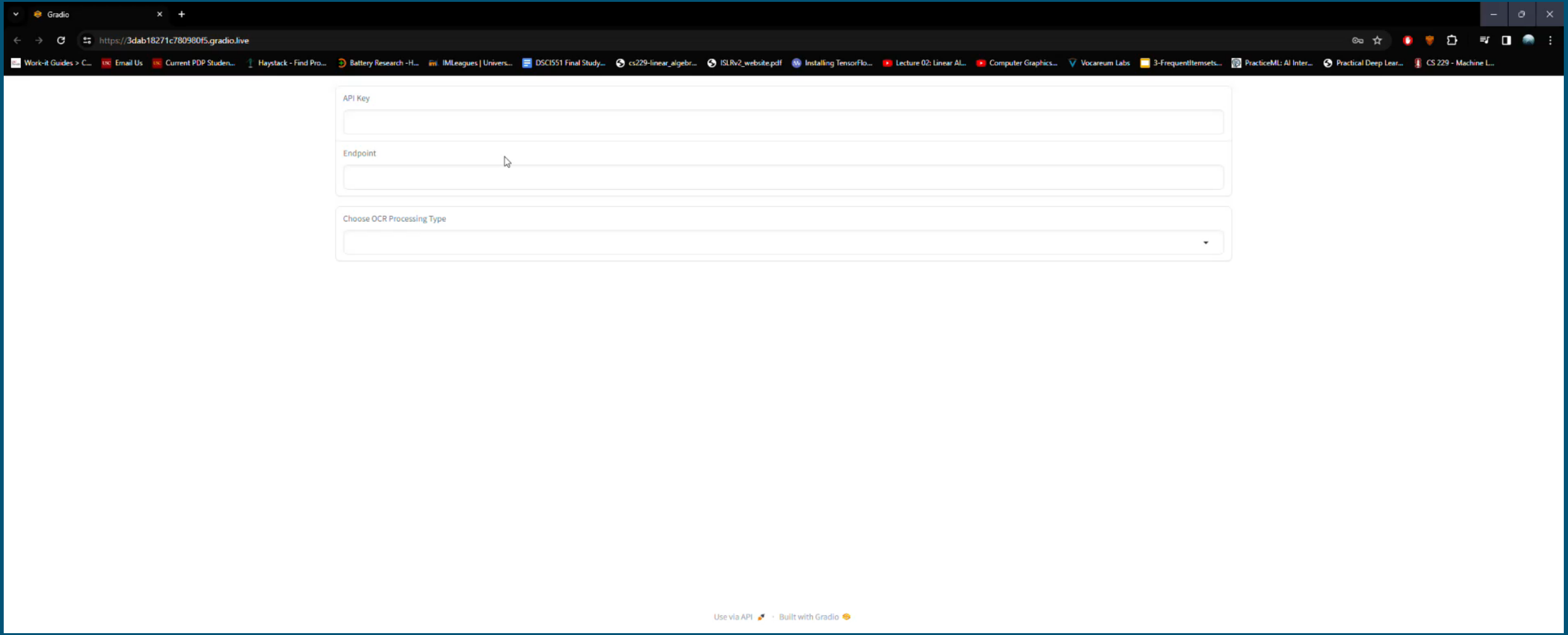
Introduction: What is Optical Character Recognition?

- Typed, printed, or handwritten text from an image → machine-readable text using AI
- Feature a two-step deep learning process: **text detection** and **text recognition**



Current OCR Capabilities

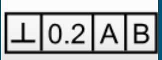
- Created a GUI using Gradio in Python for Out-of-the-box model, Azure OCR



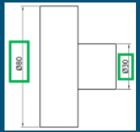
- As seen in the video, Azure OCR generates good results, but it could be better…

Shortcomings of Out-of-the-Box Models

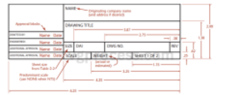
Technical Drawing Focus Areas

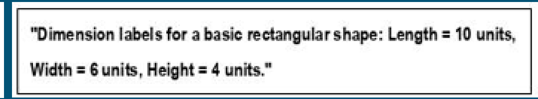
- Tolerances:
  - Often contain special characters 

- Horizontal Dimensions: 

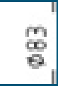
- Vertical Dimensions: 

- Description Boxes (Title blocks, General Notes, Data Box, etc.):



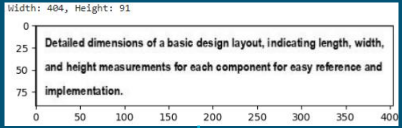


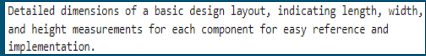
Out-of-the-Box Models

- **Detecting** areas of text on the image is exceptional (especially for textboxes), but correctly **recognizing** the text is difficult
  - Special symbols: Ø, □, ∠, ⊔, ⌓, ⊥
  - Vertical text orientation: 

Version 1

- Prioritized description box text extraction to produce metadata
  - Likely to contain keywords Sandia scientists will use for image retrieval



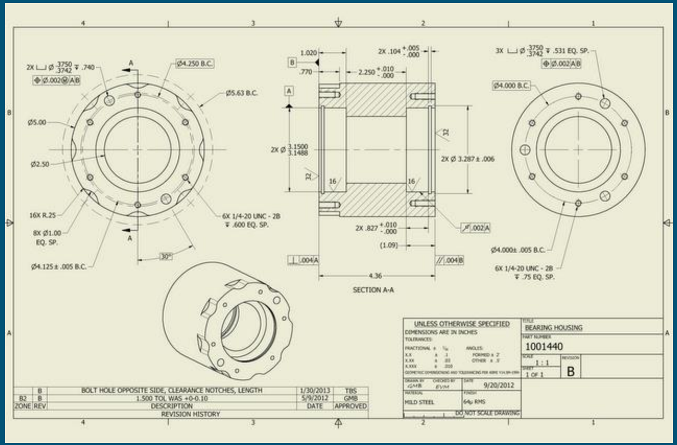


Version 2

- Prioritize correctly recognizing drawing labels
- More on this later….

Project Context

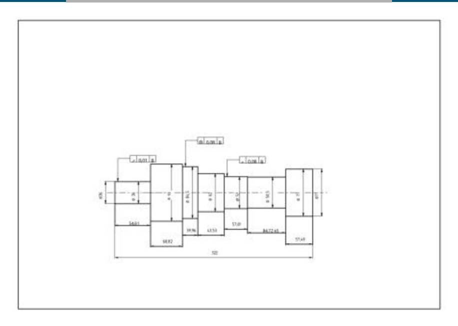
- Sandia has a database with thousands of CAD technical drawing images
- Task: Use OCR to generate metadata to speed up image retrieval for Sandia Scientists
- Technologies used: **Google Colab (Python)**



Dataset

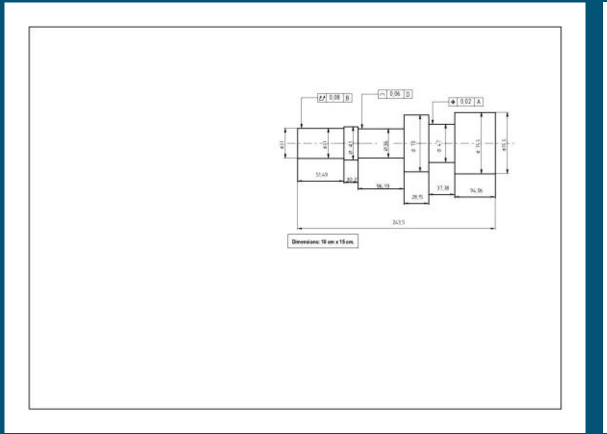
- Dilemma:
  - Cannot use Sandia's technical drawing image database
  - No publicly available technical drawing datasets online
- Found a research paper that addresses this dilemma:
  - "Text Detection on Technical Drawings for Digitization of Brown-field Processes" by Tobias Schlangenhauf, Markus Netzer, Jan Hillinger
- Used image generators from research paper
  - Generator #1: Basic technical drawing look

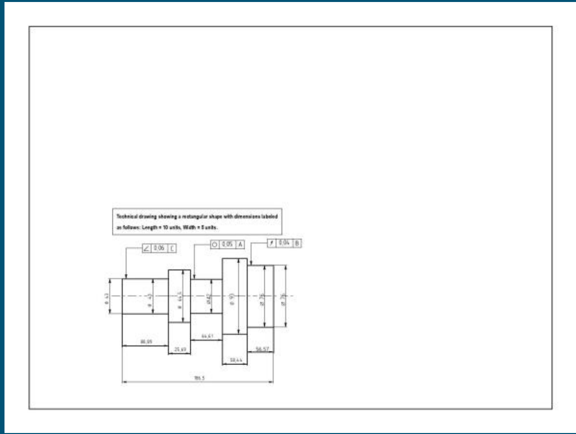
Image Generator #1 Output

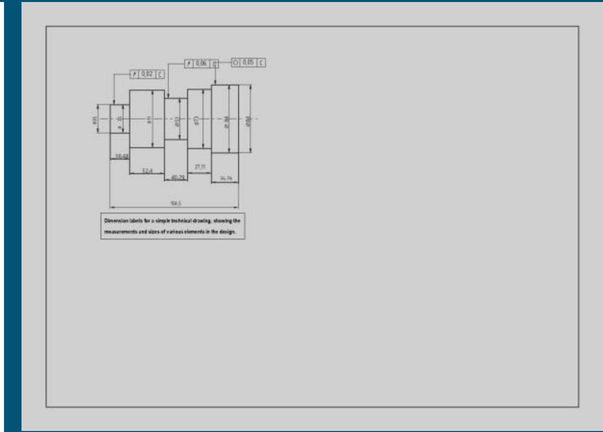


Preprocessing

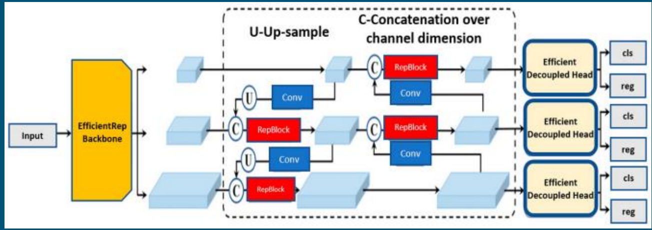
- **Preprocessing that is included in the image generator:**
  - Randomness: Polygon shape of technical drawing, size and position of geometries, values of dimensions and tolerance symbols, augmentation params (Sharpness, Brightness, Contrast), placement of the technical drawing
- **My modifications:**
  - Font size, range of geometries lengths, adjusted overall spacing/padding, changed size of bounding boxes, changed range of image dimensions, added textbox

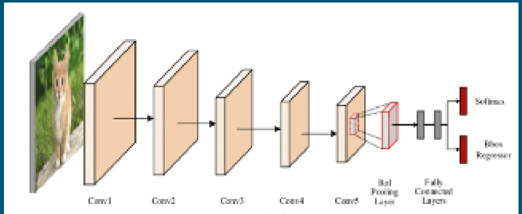






Sub-Model Selection #1: YOLOv8 (Text Detection)





<u>Why YOLOv8?</u>

<u>YOLOv8</u>

- Processes entire image in one go
- Can be easily scaled (resource efficient, multiple model sizes)
- Involves handling only one model, unified architecture = easier deployment

<u>Faster-RCNN</u>

- Two-stage process: First stage proposes regions, second stage classifies these regions and refines their boundaries
- Not easily scaled (resource intensive)
- Deployment involves managing multiple models: Region Proposal Network and classifier

Sub-Model Selection #2: Azure OCR (Text Recognition)

- Out-of-the-box model created by Microsoft
- Selected due to Sandia’s familiarity with Microsoft Azure products
- Communicate with Azure OCR via API calls to Azure Cognitive Services (Computer Vision Service)

```python
def image_to_text(image, imagename, output_path, is_pred, api_key, endpoint):
  cropped_io = io.BytesIO()
  image.save(cropped_io, format='JPEG')
  cropped_io.seek(0)
  #initialize computer vision client
  cv_client = ComputerVisionClient(endpoint, CognitiveServicesCredentials(api_key))
  #read in the image as a binary
  response = cv_client.read_in_stream(cropped_io, language = 'en', raw = True)
  #creates a unique key associated with the image
  op_location = response.headers['Operation-Location']
  #grab the unique key
  op_id = op_location.split('/')[-1]
  time.sleep(1)
  #get the result
  op_result = cv_client.get_read_result(op_id)
  #check if the result is ready
  if op_result.status == OperationStatusCodes.succeeded:
    results_read = op_result.analyze_result.read_results
    #read each result line
    for result in results_read:
      for line in result.lines:
        line_text = line.text
        print(line_text)
        file_suffix = '_pred_output.txt' if is_pred else '_gt_output.txt'
        file_path = os.path.join(output_path, imagename.split('_')[0] + file_suffix)
        #save as a text file
        with open(file_path, 'a', encoding='utf-8') as f:
          f.write(line_text + '\n')
```

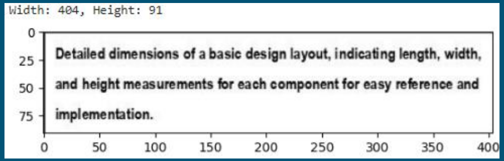
Custom OCR System Overview

1. Image is fed into YOLOv8, outputs bounding box coordinates predictions
2. Images are cropped based on bounding box coordinates and saved
3. Cropped images used as input Azure OCR
4. Azure OCR outputs recognized text in machine-readable format, saves to text file

YOLOv8 Dictionary Output


```
{'0024': [array([[     576.71,      626.44,      398.53,      62.231],
       [     821.93,      267.52,      87.789,      24.431],
       [     991.52,       337.8,      87.689,      24.519],
       [      648.3,      311.83,      86.474,      25.817],
       [     643.75,      420.33,      26.284,      54.853],
       [     728.79,      420.33,      25.784,      48.453],
       [     791.65,      421.61,      27.495,      38.831],
       [     1043.6,      425.36,      21.901,       33.11],
       [     548.43,      419.07,       28.06,      47.997],
       [     886.54,      420.94,        26.6,      44.652],
       [     979.75,      424.08,      22.994,       37.63],
       [     877.47,       543.7,       30.65,      15.729],
       [      792.5,      551.91,       32.32,      18.333],
       [      811.5,      586.93,      20.349,      16.019],
       [     638.23,      520.86,      53.898,      18.043],
       [     725.23,      587.74,       31.53,      17.626],
       [     981.06,      520.06,      29.576,       17.43]], dtype=float32)],
```


Azure OCR Input



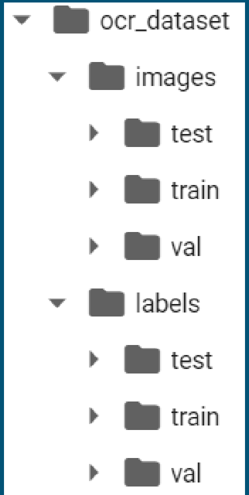
→

Azure OCR Output

```
Detailed dimensions of a basic design layout, indicating length, width,
and height measurements for each component for easy reference and
implementation.
```

Training, Validation and Test Sets

- 500 preprocessed images are randomly split into the train, validation, and test sets: 80%, 10%, and 10% respectively
  - Corresponding text file with ground truth bounding boxes for each image file saved under the labels folder
- Used to train, validate, and test the text detection (YOLOv8) model ONLY



First Stage: Transfer Learning for YOLOv8

- Goal for YOLOv8: detect all areas of the image that have text
- Transfer learning: retraining the final layers of a pre-trained model with new data
  - Train YOLOv8 (freeze first 10 layers) on training and validation images and labels
  - Predict on test images, save in dictionary, compare against testing labels

```python
model = YoloWrapper('medium')
model.train_yolo(config_file, num_epochs=100, results_name='ocr', val=True, single_cls=True, patience=30)
```

```
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
Downloading https://github.com/ultralytics/assets/releases/download/v8.1.0/yolov8n.pt to 'yolov8n.pt'...
100%|██████████| 6.23M/6.23M [00:00<00:00, 23.5MB/s]
AMP: checks passed ✅
train: Scanning /content/drive/MyDrive/ocr_dataset/labels/train... 80 images, 0 backgrounds, 0 corrupt: 100%|██████████| 80/80 [00:00<00:00, 253.85it/s]

albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01), CLAHE(p=0.01, clip_limit=(1, 4.0), tile_grid_siz
val: Scanning /content/drive/MyDrive/ocr_dataset/labels/val... 10 images, 0 backgrounds, 0 corrupt: 100%|██████████| 10/10 [00:00<00:00, 50.40it/s]val:

Plotting labels to runs/detect/ocr/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.00125, momentum=0.9) with parameter groups 77 weight(decay=0.0), 84 weight(decay=0.0005), 83 bias(decay=0.0)
TensorBoard: model graph visualization added ✅
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to runs/detect/ocr
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      3.21G      2.611      4.804       1.21        540        640: 100%|██████████| 5/5 [00:12<00:00,  2.40s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:01<00:00,  1.34s/it]
                   all         10        194     0.0191     0.0103    0.00274    0.00136


      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/100      3.19G      2.292      3.291      1.111        467        640: 100%|██████████| 5/5 [00:01<00:00,  2.91it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  5.23it/s]
```

```python
bb_dict = {}
for test_image in os.listdir(preds_path):
    image_path = os.path.join(preds_path, test_image)
    image_id = os.path.basename(test_image).split('_')[0]
    prediction = model.predict(image_path, 0.25)
    bb_dict[image_id] = prediction
```

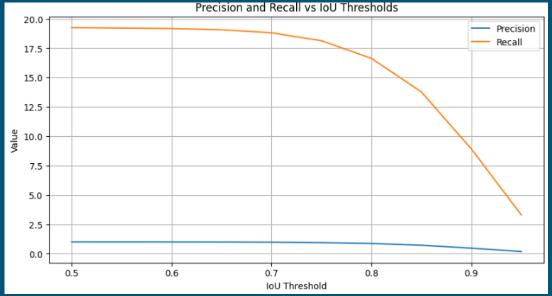
YOLOv8 Results

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{IoU} = \frac{(Object \cap Detected\ box)}{(Object \cup Detected\ box)}$$

- Average Intersection over Union (IoU) = **<u>0.8823</u>**:
  - IoU = measurement of how close a predicted bounding box is to its ground truth bounding box



- Mean Average Precision (mAP50) = **<u>0.9939</u>**:
  - Calculates average precision across all classes at an IoU threshold of 0.5

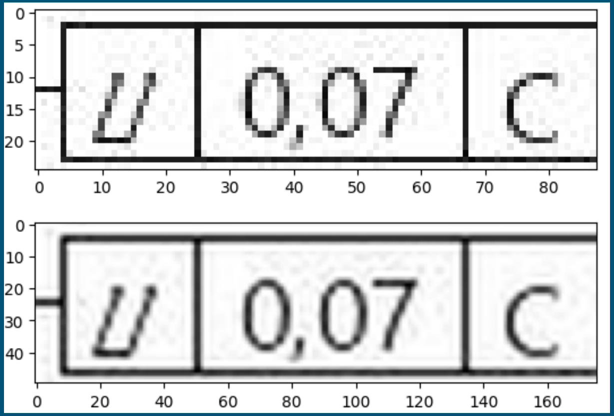
- Mean Average Precision (mAP50-95) = **<u>0.8151</u>**:
  - Average precision for each class across at each threshold 0.50 to 0.95 in increments of 0.05

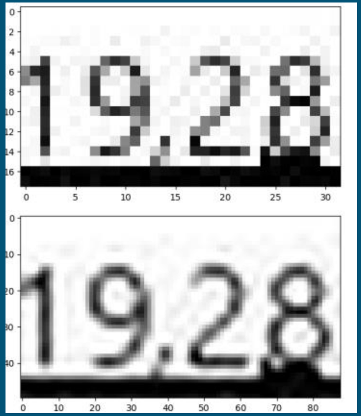
Second Stage: Azure OCR Pre-processing

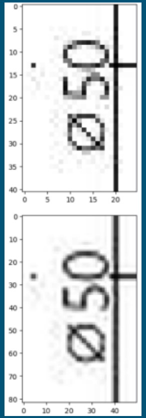
- Crop images from YOLOv8 bounding box output
- Upscaled cropped images using Image.LANCZOS to be at least 50x50 → input into Azure OCR
- Since the cropped image is so small, it needs to be upscaled
  - Upscaling makes the image more pixelated and harder to read
  - Use LANCZOS to maintain sharpness and clarity in the resized image

Before Lanczos

After Lanczos

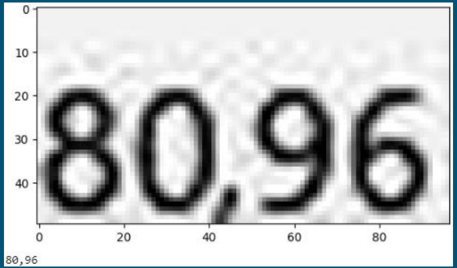


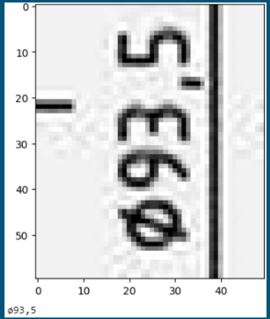


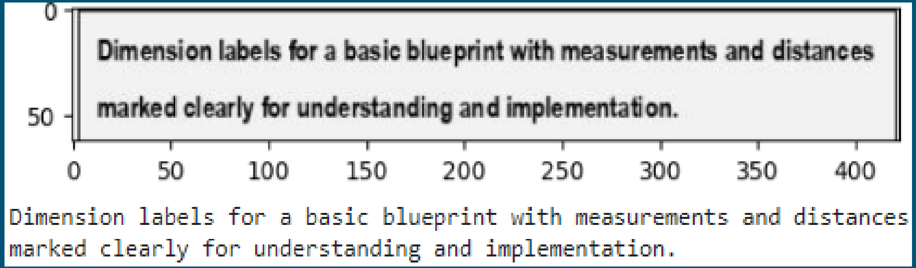


Azure OCR Results

- Accuracy measured using Character Error Rate (CER)
  - CER = (Levenshtein distance / total characters in ground truth) = 0.288 * 100 = 28.8%
  - Levenshtein distance is minimum number of single-character edits necessary to change one string to the other
- Care most about the text recognition for the textboxes
  - CER for textboxes = 0.077, that's an accuracy of .923 or 92.3%!







Conclusion

- Text Detection YOLOv8 performed very well despite “small” dataset
  - Achieves an IoU of **0.8823**, a mAP50 of **0.9939**, mAP50-95 of **0.8151**
- Text Recognition Azure OCR performed well on text recognition
  - Achieves a CER of **0.288** when recognizing text from all classes
  - Achieves a CER of **0.077** when recognizing textbox text

Future Plans: Version 1 vs Version 2

<u>Key Differences (Version 1):</u>

- Focus: Text Description Boxes
- Used Image Generator #1
- 500 images for training, validating and testing YOLOv8
- Out-of-the-box Azure OCR for Text Recognition

<u>Version 1:</u>

- Text Detection: YOLOv8 (medium size)
  - IoU: 0.8823, mAP50: **<u>0.9939</u>,** mAP50-95: 0.8151
- Text Recognition: Azure OCR
  - CER: **<u>0.288</u>**

<u>Key Differences (Version 2)</u>

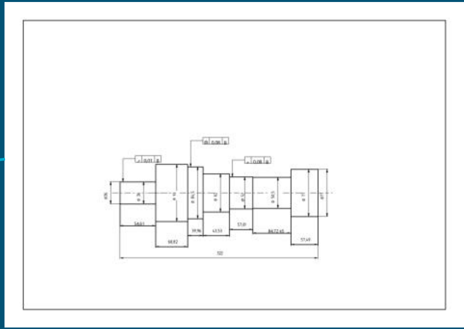
- Focus: Dimension Labels
- Used all three Image Generators
- 1000 images for training, validating and testing YOLOv11
- Fine-tuned Qwen-2 for Text Recognition
- 2400 images for training and testing TrOCR

<u>Version 2:</u>

- Text Detection: YOLOv11 (large size)
  - IoU: **<u>0.8939</u>**, mAP50: 0.9855**,** mAP50-95: **<u>0.8708</u>**
- Text Recognition: Transfer Learning with TrOCR
  - CER: .5503 (currently)

Dataset

Image Generator #1 Output



- Used image generators from research paper
  - Generator #1: Basic technical drawing look (700 images)
    - Purpose: **Structural familiarity**
  - Generator #2: Complex look (300 Images)
    - Purpose: **Differentiating text and background noise**
    - Added the different elements
  - Generator #3: Text Snippets (2400 Images)
    - Purpose: **Orientation/symbols**
    - Added: background noise, image rotation

Image Generator #2 Output

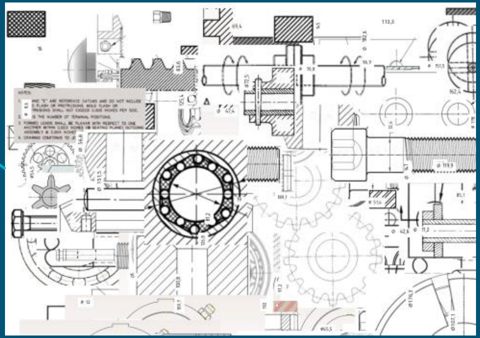
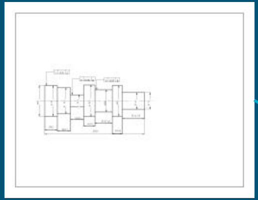


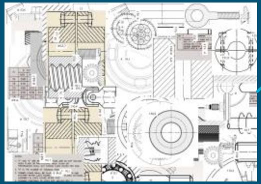
Image Generator #3 Outputs

Version System Overview

1. After training, 100 test images → YOLOv11 → bounding boxes of all areas of text from each image
2. Bounding boxes used to create cropped images
3. Cropped images → TrOCR → machine readable text
4. Machine-readable text saved to a text file with same name as the image

1. YOLOv11 Input





2. YOLOv11 Output


```
{'0024': [array([[     576.71,      626.44,      398.53,      62.231],
       [     821.93,      267.52,      87.789,      24.431],
       [     991.52,       337.8,      87.689,      24.519],
       [      648.3,      311.83,      86.474,      25.817],
       [     643.75,      420.33,      26.284,      54.853],
       [     728.79,      420.33,      25.784,      48.453],
       [     791.65,      421.61,      27.495,      38.831],
       [     1043.6,      425.36,      21.901,       33.11],
       [     548.43,      419.07,       28.06,      47.997],
       [     886.54,      420.94,        26.6,      44.652],
       [     979.75,      424.08,      22.994,       37.63],
       [     877.47,       543.7,       30.65,      15.729],
       [      792.5,      551.91,       32.32,      18.333],
       [      811.5,      586.93,      20.349,      16.019],
       [     638.23,      520.86,      53.898,      18.043],
       [     725.23,      587.74,       31.53,      17.626],
       [     981.06,      520.06,      29.576,       17.43]], dtype=float32)],
```


3a. TrOCR Input

3b. TrOCR Output

18,7

4. Text File

000106.txt

```
18,7
138,5
160,6
138,2
38,2
49,7
ø52,2
ø137,1
11,9
ø32,9
ø16,5
150
ø130
5,7
ø±7
100.,7
5,9
ø154
161
135,3
183,8
ø13±
ø101,1
192,3
ø89,6
ø89,8
ø73,2
12,4
69,3
135,8
ø58
50,9
ø191,3
ø75,8
ø61,8
ø98
ø184,5
ø39,8
ø32
11,7
```

Lessons Learned

- Most challenging parts:
  - Dataset creation
  - Model selection

- What I learned/found helpful:
  - Don’t just think about it, try it!
  - Be proactive, especially when you need help
  - Research what works, but don’t be afraid to make your own approach
  - Stay calm and try to work out big issues

Reference

- Schlagenhauf, Tobias, et al. “Text Detection on Technical Drawings for the Digitization of Brown-Field Processes.” *Procedia CIRP*, Elsevier, 18 July 2023, www.sciencedirect.com/science/article/pii/S2212827123002883.

Extra Slides

Training, Validation and Test Sets

<u>Text Detection</u>

- 1,000 preprocessed images randomly split into the train, validation, and test sets: 80%, 10%, 10%
    - The ground truth bounding box labels for each image file saved under the labels folder
- Image Generators 1 (70%) and Image Generator 2 (30%) used to train Text Detection Model
    - Image Generator 1 = For detecting text against basic technical drawing layout
    - Image Generator 2 = For detecting text against noisy backgrounds

<u>Text Recognition</u>

- 1,500 preprocessed images randomly split into the train, validation, and test sets: 80%, 10%, 10%
- Only Image Generator 3
    - For recognizing special technical drawing symbols/labels

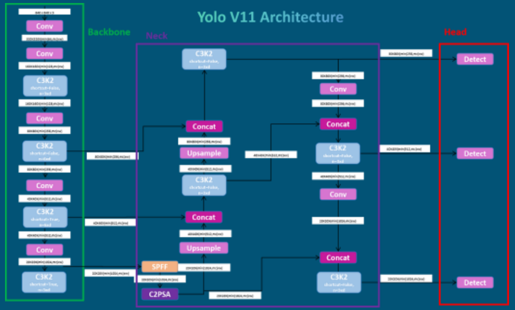
Preprocessing

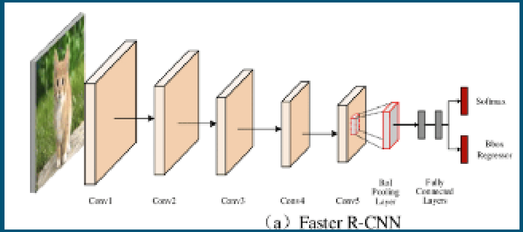
- Image Generator #1 Preprocessing
  - Random geometries, random text generation, several augmentations (sharpness, brightness, contrast), bounding box generation
  - I added: Padding and Cropping Adjustments, output directories, bounding box creation, output directories

- Image Generator #2 Preprocessing
  - Image size variation, random element overlay, random text placement and dimension values
  - I added: Input 80 technical drawing element snippets, bounding box creation, output directories

- Image Generator #3 Preprocessing
  - Random text creation and font selection, Image generation and text placement
  - I added: Random Image Rotation, kept track of bounding boxes, background noise (line, circle, rectangle)

Sub-Model #1: YOLOv11 (Text Detection)





Why YOLOv11?

YOLOv11
- Processes entire image in one go
- Can be easily scaled (resource efficient, multiple model sizes)
- Involves handling only one model, unified architecture = easier deployment

Faster-RCNN
- Two-stage process: First stage proposes regions, second stage classifies these regions and refines their boundaries
- Not easily scaled (resource intensive)
- Deployment involves managing multiple models: Region Proposal Network and classifier

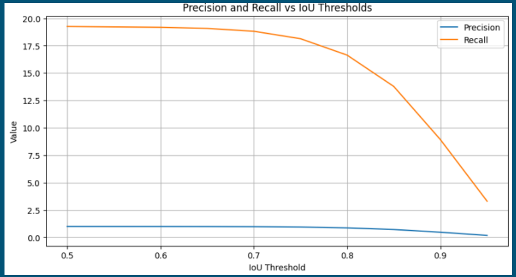
YOLOv11 Results

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{IoU} = \frac{(Object \cap Detected\ box)}{(Object \cup Detected\ box)}$$

- Average Intersection over Union (IoU) = **<u>.8924</u>**:
  - IoU = measurement of how close a predicted bounding box is to its ground truth bounding box
- Mean Average Precision (mAP50) = **<u>0.9774</u>**:
  - Calculates average precision across all classes at an IoU threshold of 0.5
- Mean Average Precision (mAP50-95) = **<u>0.8606</u>**:
  - Average precision for each class at threshold 0.50 to 0.95 in increments of 0.05



First Stage: Transfer Learning for YOLOv11

- Goal for YOLOv8: detect all areas of the image that have text
- Transfer learning: retraining the final layers of a pre-trained model with new data
  - Train YOLOv11 (freeze first 10 layers) on training and validation images and labels (bounding boxes)
  - Predict the bounding boxes for the test images:

```
model = YOLO('yolo11l.pt')
model.train(data=config_file, epochs=100, name='ocr', val=True, single_cls=True, patience=10)
```

```
Transferred 1009/1015 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/ocr2', view at http://localhost:6006/
Freezing layer 'model.23.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks...
AMP: checks passed ✅
train: Scanning /content/drive/MyDrive/ocr_v2_dataset/labels/train.cache... 800 images, 0 backgrounds, 0 corrupt: 100%|██████████| 800/8
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, num_output_channels=3, method='we

val: Scanning /content/drive/MyDrive/ocr_v2_dataset/labels/val.cache... 100 images, 0 backgrounds, 0 corrupt: 100%|██████████| 100/100 [
Plotting labels to runs/detect/ocr2/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' autom
optimizer: AdamW(lr=0.001429, momentum=0.9) with parameter groups 167 weight(decay=0.0), 174 weight(decay=0.0005), 173 bias(decay=0.0)
TensorBoard: model graph visualization added ✅
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to runs/detect/ocr2
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      11.5G       2.07      1.705      1.082        543        640: 100%|██████████| 50/50 [00:38<00:00,  1.29it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.62it/s]


      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/100      11.1G      1.586     0.7982      0.906        504        640: 100%|██████████| 50/50 [00:35<00:00,  1.40it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.65it/s]


      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      3/100      11.3G      1.497     0.7617     0.8918        568        640: 100%|██████████| 50/50 [00:35<00:00,  1.41it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:01<00:00,  2.68it/s]
```

Important Parameters

Sub-Model #2: TrOCR (Text Recognition)

- Transformer-based OCR Pre-trained Model
- Selected because of it can be fine-tuned via Transfer Learning
- Uses transformer architecture for image understanding and text generation