Qualcomm: Revolutionizing Industries w/ Computer Vision & Vision Language Models

AI Studio Final Presentation

University of California, Los Angeles
December 4, 2024

Team Introductions

Meet Our Team!

Jaydee Nguyen
CSULB

Phi Hung Nguyen
UCLA

Annie Cen
UCLA

Parnian Ghapandar Kashani
UCLA

Our AI Studio TA and Challenge Advisors

Muskan Rizwan Shaikh
AI Studio TA

Aastha Singh
Challenge Advisor

Presentation Agenda

1. Project Overview
2. Course of Action
3. Data
   a. Understanding
   b. Processing
4. Model
   a. Understanding
   b. Selection
   c. Evaluation
5. Final Thoughts
6. Questions

AI Studio Project Overview

Our objective is to develop and fine-tune a deep learning, computer vision model that is capable of quickly and accurately identifying weapons in public spaces.

Business Impact

- Assists in quickly identifying potential threats in real time.
- Reduces reaction and response time.
  - Without AI, it takes longer to identify positive threats and deploy safety reinforcements.
    - It takes a minimum of 8 minutes for police to arrive at an active shooter location.

- How?
  - Invisible to the perpetrator.
  - Utilizing AI eliminates the human factor in dangerous situations.

- Benefits public safety and versatile in many environments.
  - Recreational
  - Educational
  - Residential
  - Businesses

Course of Action

Our Approach

SEPT

Team Planning

- Establishing team expectations.
- Meeting arrangements.
- Narrowing project scope.

OCT

Research & Data Understanding

- Researching interested business use case.
- Searching for suitable datasets.
- Understanding necessary softwares.

NOV

Data Preparation & Modeling

- Data preparation.
- Learning Florence-2.
- Creating and building model.

DEC

Model Evaluation & Project Finalization

- Fine-tuning model.
- Finishing touches.
- Final thoughts & presentation.

Some Resources We Leveraged

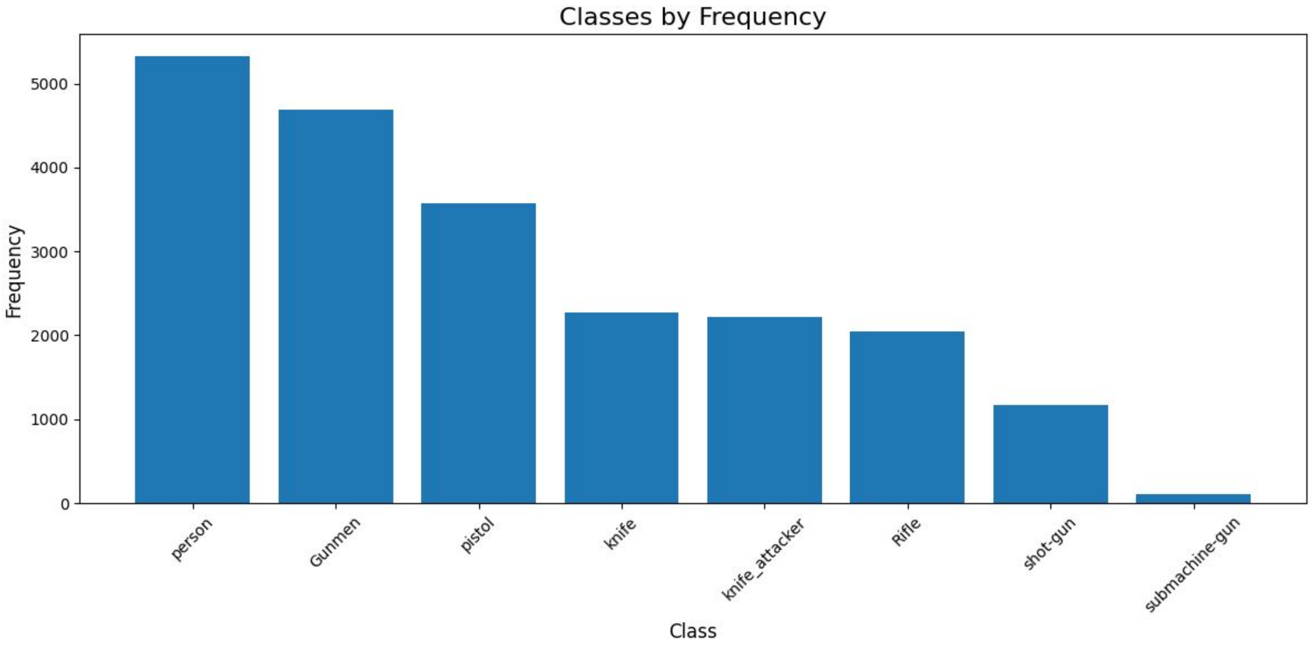
- Language
  - Python
- IDE
  - Google CoLab
- Frameworks
  - Pytorch
- Model
  - Florence-2
- Libraries
  - RoboFlow
  - HuggingFace
    - Transformers
    - Timm
    - PEFT
  - Einops

Data Processing & Understanding

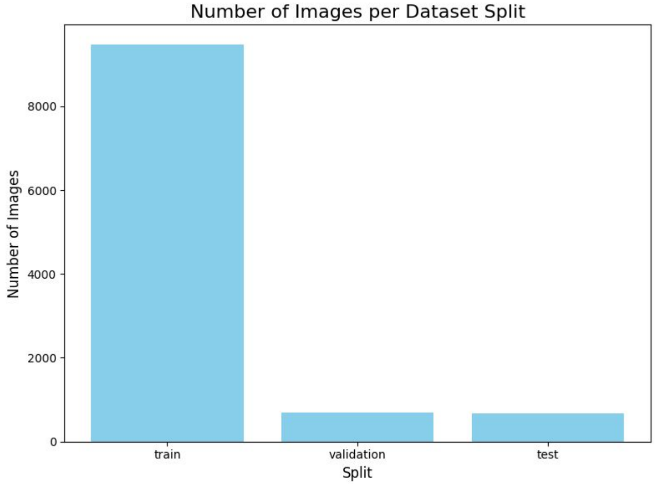
Dataset Analytics

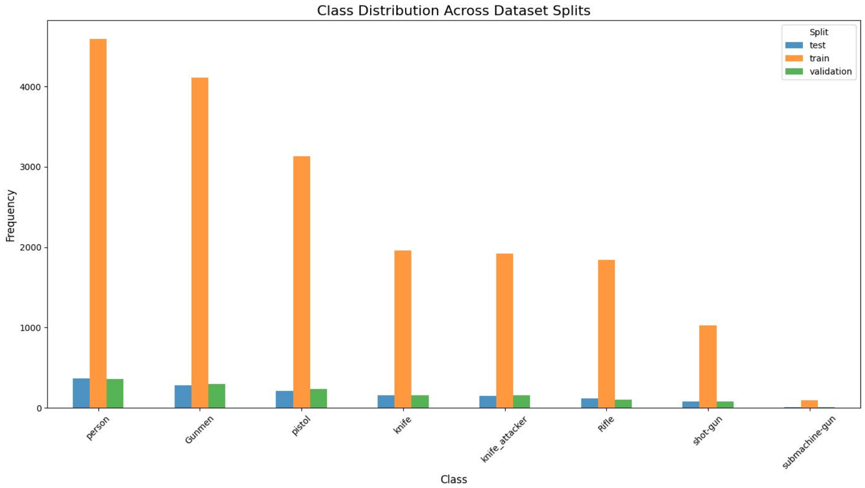
10000+ Images

8 Classes



Dataset Visualizations





- **88% Train**
- **6% Validation**
- **6% Test**

Submachine-gun is greatly underrepresented

Dataset Import and Sampling

Data import: utilize the Roboflow API

```
ROBOFLOW_API_KEY = userdata.get('ROBOFLOW_API_KEY')
rf = Roboflow(api_key=ROBOFLOW_API_KEY)
project = rf.workspace("weapon-detect-qbsiw").project("yolo-weapon-detection")
version = project.version(9)
dataset = version.download("florence2-od")
```

Data Sampling and Rebalancing

- **Purpose**
  - To create manageable subsets and balances computational efficiency while ensuring each split retains representative distribution
- **Method**
  - Random Sampling - Using 10% fixed fraction of annotations randomly selected from each subset
  - Class Filtering - Retained rare classes like “Submachine-gun” to avoid underrepresentation when sampling

```
# base_dir = "Weapon-Detection-2"
base_dir=dataset.location
train_dir = os.path.join(base_dir, "train")
valid_dir = os.path.join(base_dir, "valid")
test_dir = os.path.join(base_dir, "test")

# Output directories for subsets
output_dir = "Weapon-Detection-Subset"
os.makedirs(output_dir, exist_ok=True)

train_subset_dir = os.path.join(output_dir, "train")
valid_subset_dir = os.path.join(output_dir, "valid")
test_subset_dir = os.path.join(output_dir, "test")

os.makedirs(train_subset_dir, exist_ok=True)
os.makedirs(valid_subset_dir, exist_ok=True)
os.makedirs(test_subset_dir, exist_ok=True)

# Function to sample the dataset and create JSONL files
def sample_subset(input_dir, jsonl_path, output_dir, subset_jsonl_path, fraction=0.1):
    # Load the original annotations
    with open(jsonl_path, 'r') as f:
        annotations = [json.loads(line) for line in f]

    # Sample annotations
    sampled_annotations = random.sample(annotations, int(len(annotations) * fraction))
    #classes with fewer examples should be kept
    for annotation in annotations:
      if(detect_rare_classes(annotation) and annotation not in sampled_annotations):
        sampled_annotations.append(annotation)

    # Copy sampled images and create a new JSONL file
    os.makedirs(output_dir, exist_ok=True)
    with open(subset_jsonl_path, 'w') as f:
        for annotation in sampled_annotations:
            # Copy the corresponding image
            image_path = os.path.join(input_dir, annotation['image'])
            if os.path.exists(image_path):
                shutil.copy(image_path, os.path.join(output_dir, annotation['image']))
                # Write the annotation to the new JSONL file
                f.write(json.dumps(annotation) + '\n')
```

Dataset inspection after processing

Examining Data Distribution

- **Purpose**
  - To analyze the dataset's class distribution and identify imbalances
  - Understanding the class representation helps us evaluate further steps
- **Method**
  - Processed each image and its annotations
  - Counted occurrences of each class and visualized the results
  - Identified instances with missing annotations and explored distribution

```python
### examine the ditribution of dataset
import matplotlib.pyplot as plt
task='<OD>'
text='<OD>'
class_distribution={"None":0}
for image,data in train_dataset.dataset:
    target = processor.post_process_generation(data["suffix"], task='<OD>', image_size=image.size)
    target_detections = sv.Detections.from_lmm(sv.LMM.FLORENCE_2, target, resolution_wh=image.size)
    class_names=target_detections["class_name"]
    if class_names is None:
      print("suffix is",data["suffix"])
      class_distribution["None"]+=1
      bounding_box_annotator = sv.BoxAnnotator(color_lookup=sv.ColorLookup.INDEX)
      label_annotator = sv.LabelAnnotator(color_lookup=sv.ColorLookup.INDEX)

      image = bounding_box_annotator.annotate(image, target_detections)
      image = label_annotator.annotate(image, target_detections)
      image.thumbnail((600, 600))
      plt.imshow(image)
      plt.axis("off")  # Hide axes for better viewing
      plt.show()

    else:
      for class_name in class_names:
        if class_name in class_distribution.keys():
          class_distribution[class_name]+=1
        else:
          class_distribution[class_name]=1
```

- Result: `{'None': 95, 'person': 489, 'Gunmen': 448, 'shot-gun': 106, 'Rifle': 214, 'pistol': 297, 'knife_attacker': 207, 'knife': 211, 'submachine-gun': 90}`

Insights & Key Findings

Data Sampling

- Training, Validation, Testing: ~ 10% of the original training
- These subsets reduced data size while improving class balance in each split.

Examining Data Distribution

- Most Frequent Classes
  - Person, Gunmen, Pistol
- Rare Classes
  - Submachine-gun, Shotgun, Knife_attacker

Next-up: Modeling and Evaluation

Modeling Process & Evaluation

Model Selection

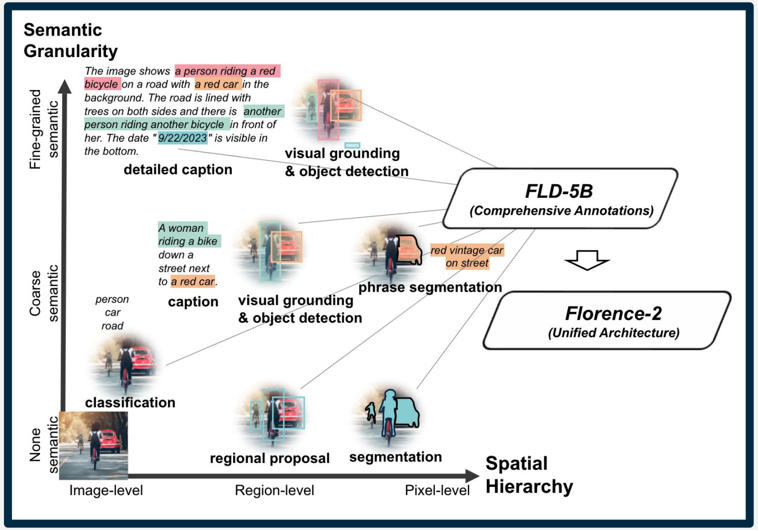
- Problem
  - Object Detection
    - Multiclassification
      - Supervised
- Chosen Model
  - LoRA Florence-2

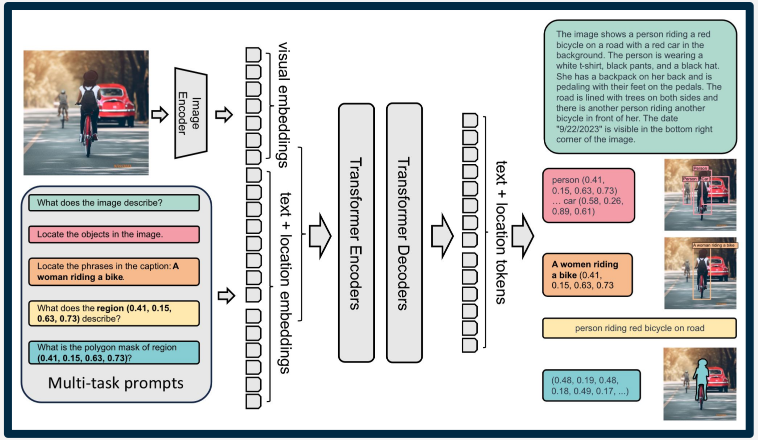
- Classes
  - Gunmen
  - Submachine-gun
  - Knife-Attacker
  - Knife
  - Rifle
  - Person
  - Pistol
  - Shotgun

Florence-2

Key features and Architectures

- Combines DaViT vision encoder with BERT text embeddings.
- Transformer-based multi-modal encoder-decoder.
- Supports object detection, captioning, grounding, and segmentation.
- Excels in zero-shot and fine-tuned tasks.





Florence-2 Architecture

Vision Encoder

- **DaViT (Dynamic Attention Vision Transformer)**:
  - Processes an input image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$
  - Converts the image into flattened **visual token embeddings** $\mathbf{V} \in \mathbb{R}^{N_v \times D_v}$

Multi-Modality Encoder-Decoder

- **Text Embeddings**:
  - Obtain text embeddings of the form $\mathbf{T}_{prompt} \in \mathbf{R}^{N_t \times D}$ using language tokenizers and word embedding layers.
- **Dimensionality Alignment**:
  - Vision embeddings **V** are projected and normalized into $\mathbf{V}' \in \mathbb{R}^{N_v \times D}$ for compatibility with text embeddings.
- **Combined Input:**

$$\mathbf{X} = [\mathbf{V}', \mathbf{T}_{prompt}]$$

Florence-2, Universal Backbone

- **Universal Backbone**: Unified architecture with prompt-based representation for task-specific flexibility.The term **prompt-based representation** means that the model can take a specific "instruction" (a prompt) to adapt its behavior for a specific task. For example:A prompt might tell the model: "Focus on detecting objects" or "Classify this image."
- **Automated Data Annotation**:
  - **FLD-5B Dataset**: 126M images with 5.4B annotations.
  - Eliminates labor-intensive manual labeling by using Data Engines.
  - Uses multiple models to collaborate and annotate, inspired by the "wisdom of crowds."
  - Refines annotations iteratively using pre-trained foundational models.

Challenges Addressed:

- Limited annotated data availability for multitask learning.
- Lack of a unified system for diverse vision tasks.

Advantages:

- **Scalable**: Generates vast, high-quality datasets efficiently.
- **Accurate**: Consensus-based annotation ensures reliability.
- **Efficient**: Single architecture replaces task-specific models, enabling broader adaptability.

Florence-2, Universal Backbone

A. Supported Tasks and Annotations in Florence-2

Task	Annotation Type	Prompt Input	Output
Caption	Text	Image, text	Text
Detailed caption	Text	Image, text	Text
More detailed caption	Text	Image, text	Text
Region proposal	Region	Image, text	Region
Object detection	Region-Text	Image, text	Text, region
Dense region caption	Region-Text	Image, text	Text, region
Phrase grounding	Text-Phrase-Region	Image, text	Text, region
Referring expression comprehension	Region-Text	Image, text	Text, region
Open vocabulary detection	Region-Text	Image, text	Text, region
Referring segmentation	Region-Text	Image, text	Text, region
Region to text	Region-Text	Image, text, region	Text
Text detection and recognition	Region-Text	Image, text	Text, region

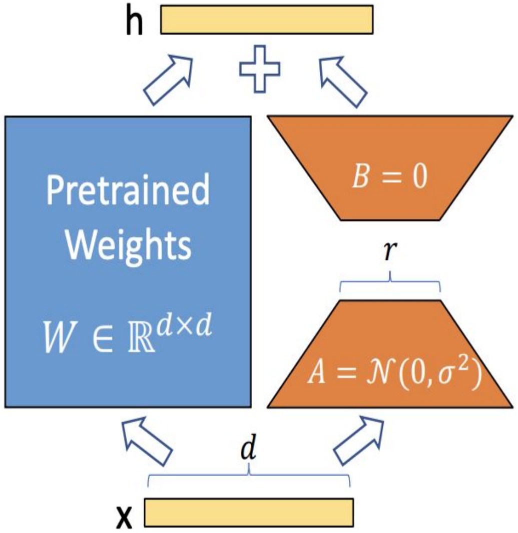
Table 13. Supported Tasks and annotations used for *Florence-2* pretraining.

Florence-2 Object Detection

```python
image = Image.open(EXAMPLE_IMAGE_PATH)
task = "<OD>"
text = "<OD>"

inputs = processor(text=text, images=image, return_tensors="pt").to(DEVICE)
generated_ids = peft_model.generate(
    input_ids=inputs["input_ids"],
    pixel_values=inputs["pixel_values"],
    max_new_tokens=1024,
    num_beams=3
)
generated_text = processor.batch_decode(generated_ids, skip_special_tokens=False)[0]
response = processor.post_process_generation(generated_text, task=task, image_size=(image.width, image.height))
detections = sv.Detections.from_lmm(sv.LMM.FLORENCE_2, response, resolution_wh=image.size)
```

Fine-Tuning with Low Rank Adaptation (LoRA)



weight matrix $W_0 \in \mathbb{R}^{d \times k}$, we constrain its update by representing the latter with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0 x$, our modified forward pass yields:

$$h = W_0 x + \Delta W x = W_0 x + BAx \tag{3}$$

Florence-2

Setup and fine-tuning

- Configure Hugging Face and Roboflow API keys.
- Use Roboflow for data preparation.
- Train with GPU resources and LoRA for efficient fine-tuning.

Training and Inference

- Train using PyTorch's AdamW optimizer and learning rate scheduler.
- Monitor training/validation losses.
- Employ confusion matrix to evaluate the model.

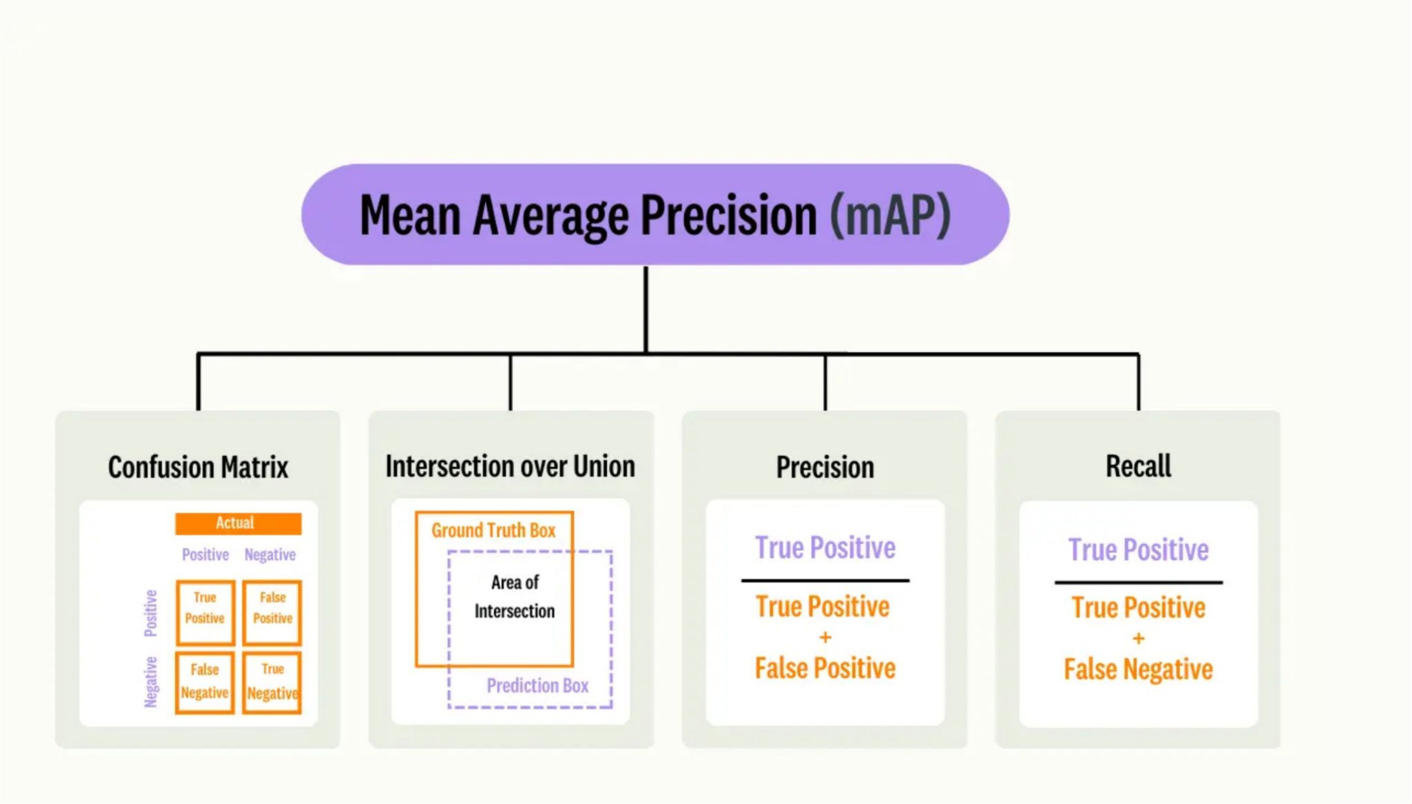
Calculate mAP

```python
# @title Calculate mAP
mean_average_precision = sv.MeanAveragePrecision.from_detections(
    predictions=predictions,
    targets=targets,
)

print(f"map50_95: {mean_average_precision.map50_95:.2f}")
print(f"map50: {mean_average_precision.map50:.2f}")
print(f"map75: {mean_average_precision.map75:.2f}")
```

```
map50_95: 0.75
map50: 0.81
map75: 0.81
```

Evaluation Metrics



Evaluation Metrics

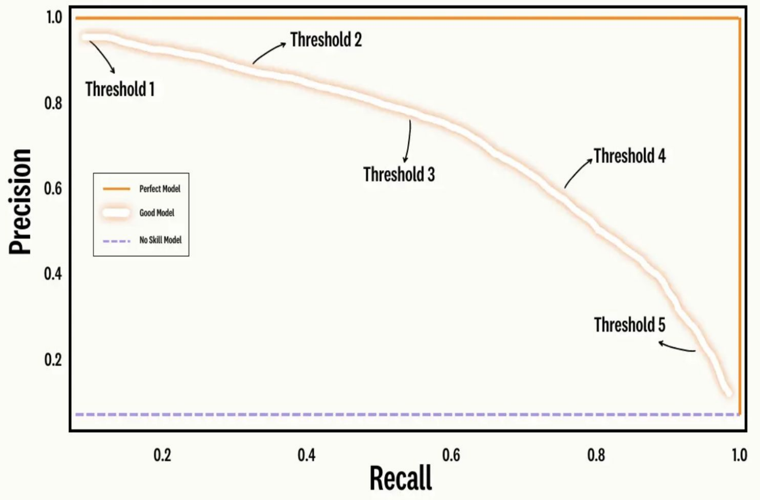
Mean Average Precision Formula

$$\text{Mean Average Precision} = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

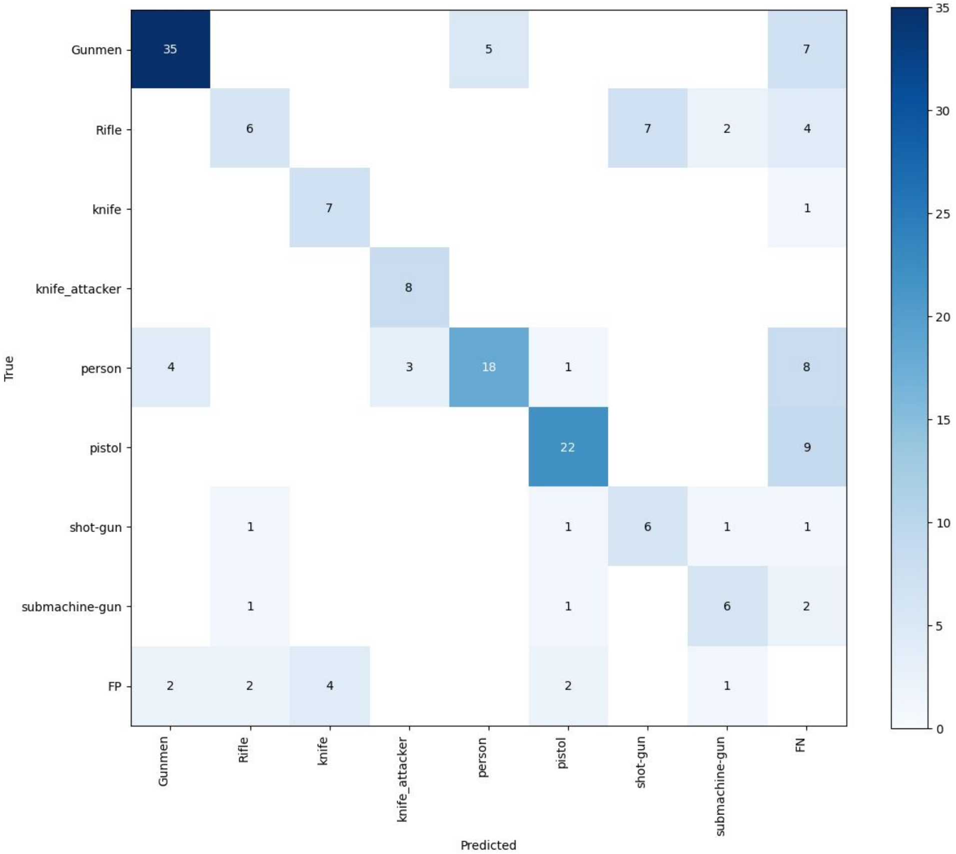
n = the number of classes

AP_k = the average precision of class k

Precision-Recall Curve



Evaluation Metrics



Calculate mAP

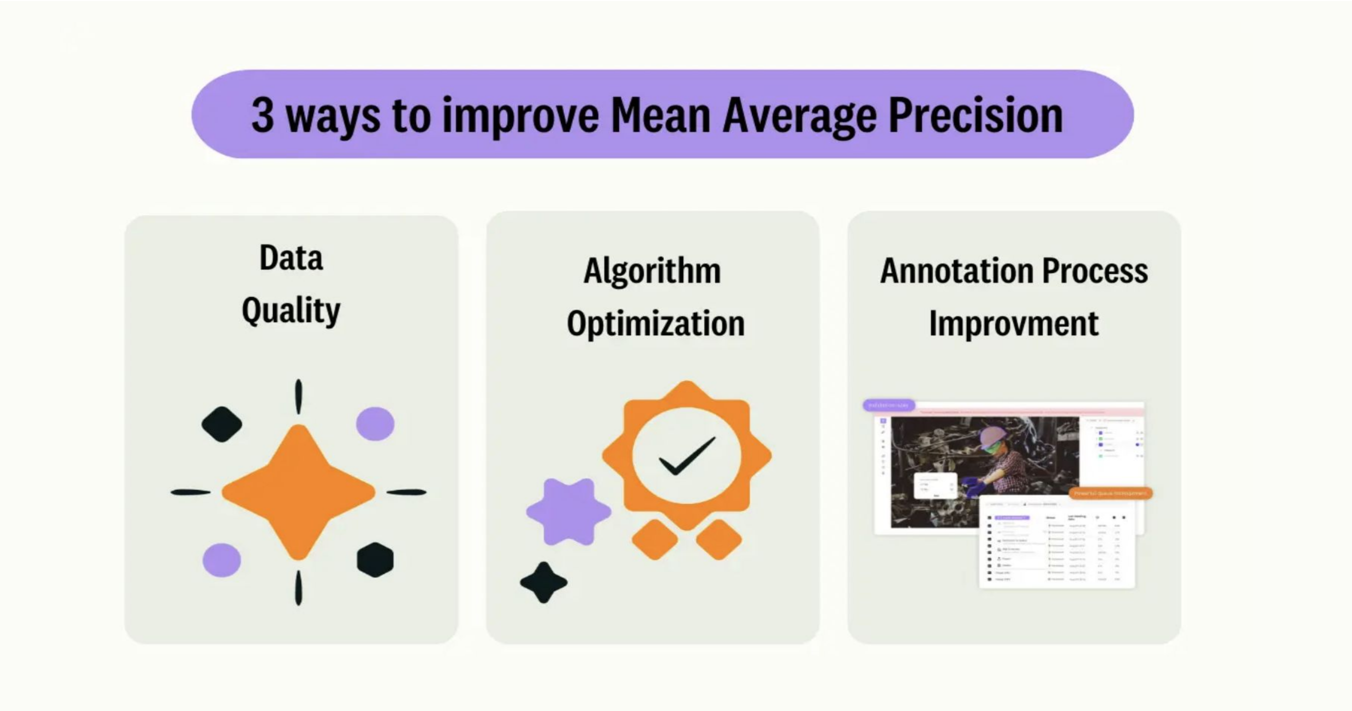
```
[ ] # @title Calculate mAP
    # Filter out any invalid predictions or targets
    # valid_predictions = [p for p in predictions if p.class_id is not None and len(p.class_id) > 0]
    # valid_targets = [t for t in targets if t.class_id is not None and len(t.class_id) > 0]

    mean_average_precision = sv.MeanAveragePrecision.from_detections(
        predictions=predictions,
        targets=targets,
    )

    print(f"map50_95: {mean_average_precision.map50_95:.2f}")
    print(f"map50: {mean_average_precision.map50:.2f}")
    print(f"map75: {mean_average_precision.map75:.2f}")
```

```
map50_95: 0.46
map50: 0.63
map75: 0.47
```

Evaluation Metrics



Insights & Key Findings

- Problems We Encountered
  - Finding a suitable multiclass dataset
    - Underfitting/Overfitting
    - Class imbalances
    - Loss of data during processing
  - Low Precision Value
- Potential Solutions
  - Continue experimenting with other datasets
  - Enhance dataset by improving quality
  - Add more diversity to dataset
  - Experiment with fine-tuning technique

Final Thoughts

What We Learned

- How to deal with RoboFlow datasets
  - Formatting
  - Analyzing
  - Sampling

- Deeper Understanding of
  - LoRA technique
  - Training for object detection

- Handling Model Errors

Obstacles

- GPU Power
- Memory Size
- Time
  - Training
  - Overall

Potential Next Steps

- Further experiment with LoRA fine-tuning technique and increase accuracy
- Attempt to scale up and train on larger sample of the dataset

Questions?

Model Comparison

Model Name	Description	Results	Pros	Cons
[Insert your text]	[Insert your text]	[Insert your text]	[Insert your text]	[Insert your text]
[Insert your text]	[Insert your text]	[Insert your text]	[Insert your text]	[Insert your text]
[Insert your text]	[Insert your text]	[Insert your text]	[Insert your text]	[Insert your text]

For Students: Blank slides to copy/paste and use as needed

Click to add title

Click to add text

Paste an image

Click to add title

Click to add text

Click to add title

Click to add text

Paste an image

Paste an image

Click to add title

Data Point

Caption providing context.

Data Point

Caption providing context.

Data Point

Caption providing context.