Perception & Computer Vision

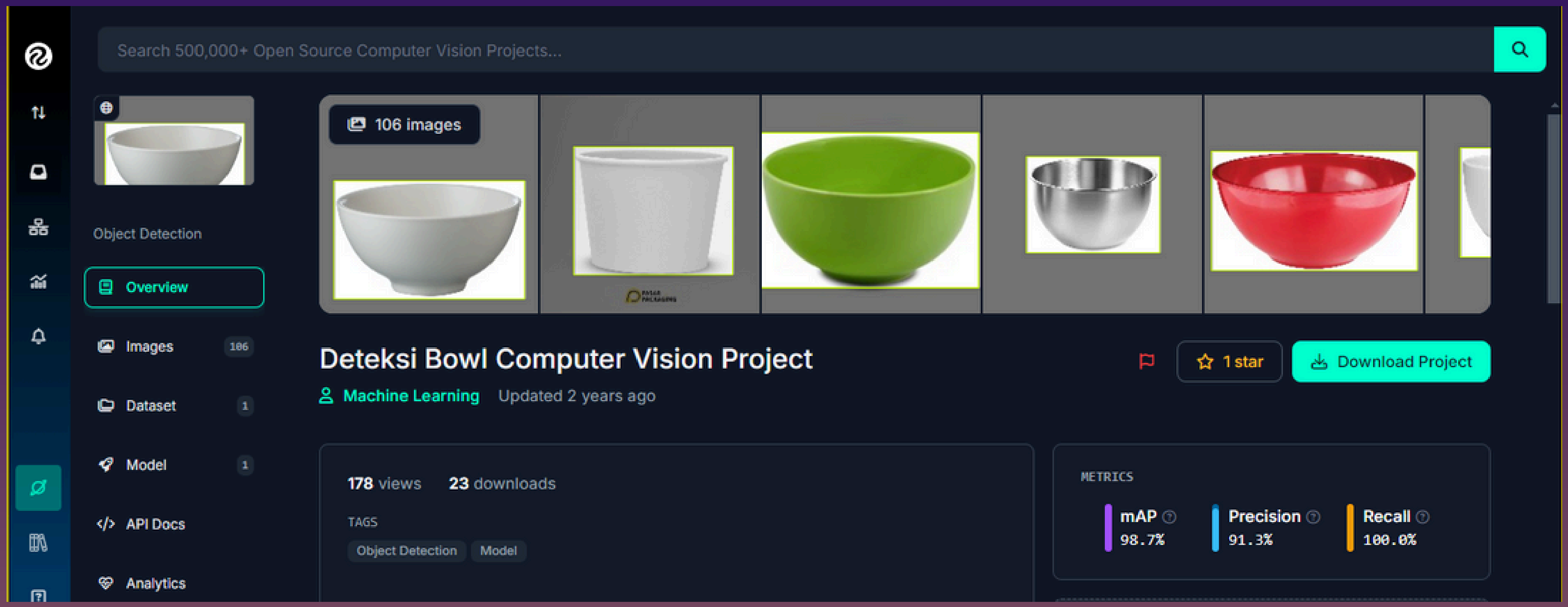
CSST 103

GAGALANG
GALZOTE
CS4A

SELECTION OF DATASET & ALGORITHM

With the usage of Histogram of Oriented Gradients with Support Vector Machine (HOG-SVM) have made the programmers workload light, as the algorithm has only extracted the features as it was being applied to the chosen dataset. It is automatically being processed as well as continuous which lessen the struggles that they may encounter. In addition, HOG-SVM doesn’t have any training model process since the chosen data is pre-trained already.

YOLO Data Preperation




```
[ ] # Install required packages
    !pip install -q ultralytics  # If YOLO model comes from ultralytics

    # Import libraries
    from ultralytics import YOLO
    import glob
    import os


    # Load the YOLO model
    yolo_model = YOLO('yolov8n.pt')

    # Load class names
    with open('/content/coco.names', 'r') as f:
        classes = [line.strip() for line in f.readlines()]

    print("Classes loaded:", classes)

    Classes loaded: ['bowl']
```

```
[ ] # Get image paths for the dataset
    image_paths = glob.glob('/content/Deteksi-Bowl-1/train/images/*.[jJ][pP][gG]')
    print("Found images:", image_paths[:5])

    Found images: ['/content/Deteksi-Bowl-1/train/images/mangkok-25-_jpg.rf.7085a3da1d3986fe8863c2aee1351e42.jpg',
```

```
[ ] # PRE-PROCESSING

    # Function to preprocess images
    def preprocess_image(image_path, target_size=(640, 640)):
        image = cv2.imread(image_path)
        if image is None:
            print(f"Error loading image: {image_path}")
            return None

        # Resize and normalize image
        image_resized = cv2.resize(image, target_size)
        image_normalized = image_resized / 255.0
        return image_normalized

    # Preprocess all images
    processed_images = [preprocess_image(path) for path in image_paths]
    print("Processed images:", len(processed_images))

    Processed images: 220
```

YOLO Model Implementation

```
[ ]  # OBJECT DETECTION

     # Function to detect objects in an image
     def detect_objects(image_path):
         image = cv2.imread(image_path)
         if image is None:
             print(f"Error loading image: {image_path}")
             return None

         image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
         results = yolo_model.predict(source=image_rgb)

         print(f"Detected {len(results)} objects in {image_path}")

         for result in results:
             for box in result.boxes:
                 x1, y1, x2, y2 = box.xyxy[0].cpu().numpy().astype(int)
                 conf = box.conf[0].cpu().numpy()
                 cls = box.cls[0].cpu().numpy()

                 if 0 <= int(cls) < len(classes):
                     label = f"{classes[int(cls)]}: {conf:.2f}"
                 else:
                     print(f"Warning: Class ID {int(cls)} not found in coco.names")
                     label = f"Class {int(cls)}: {conf:.2f}"

                 cv2.rectangle(image_rgb, (x1, y1), (x2, y2), (0, 255, 0), 3)
                 cv2.putText(image_rgb, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1)

         return image_rgb
```

YOLO Training the Model

```python
# TRAINING PARAMETERS
train_params = {
    "data": "/content/Deteksi-Bowl-1/data.yaml",
    "epochs": 10,  # Number of training epochs
    "batch": 16,   # Batch size
    "imgsz": 640,  # Image size for training
}

# Train the model
yolo_model.train(**train_params)
```

YOLO Evaluation and Testing

```
# EVALUATION OF MODEL
result = yolo_model.val(data='/content/Deteksi-Bowl-1/data.yaml')
print("Evaluation results:", result)
```

Show hidden output

```
# Run evaluation
results = yolo_model.val(data='/content/Deteksi-Bowl-1/data.yaml')

# Access and calculate mean values for evaluation metrics
precision = results.box.p.mean()  # Mean Precision across classes
recall = results.box.r.mean()     # Mean Recall across classes
map50 = results.box.map50.mean()  # Mean mAP@0.5 across classes
map50_95 = results.box.map.mean()  # Mean mAP@0.5:0.95 across classes

print("Evaluation Results:")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"mAP@0.5: {map50:.4f}")
print(f"mAP@0.5:0.95: {map50_95:.4f}")
```

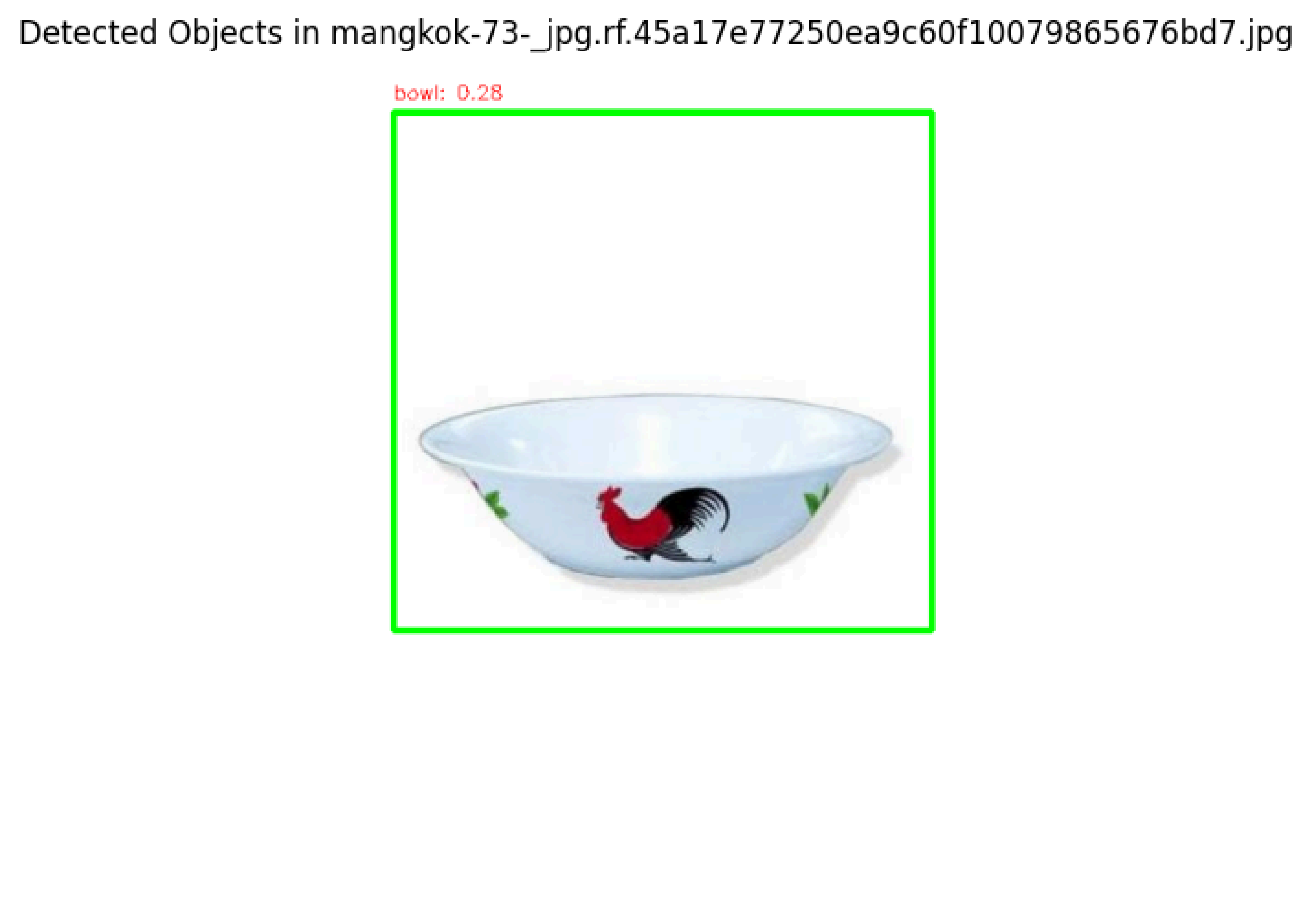
```
Ultralytics 8.3.28 🚀 Python-3.10.12 torch-2.5.0+cu121 CPU (Intel Xeon 2.20GHz)
val: Scanning /content/Deteksi-Bowl-1/valid/labels.cache... 21 images, 0 backgrounds, 0 corr
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
                   all         21         21      0.675      0.905      0.796       0.42
Speed: 3.2ms preprocess, 251.3ms inference, 0.0ms loss, 6.8ms postprocess per image
Results saved to runs/detect/train33
Evaluation Results:
Precision: 0.6749
Recall: 0.9048
mAP@0.5: 0.7959
mAP@0.5:0.95: 0.4203
```

```
import time

test_paths = glob.glob('/content/Deteksi-Bowl-1/test/images/*.jpg')
# Measure speed on sample images
start_time = time.time()
for img_path in test_paths[:4]:
    detected_image = detect_objects(img_path)
end_time = time.time()

print(f'Time taken for detection on {len(test_paths[:4])} images: {end_time - start_time:.2f} seconds')
```

```
0: 640x640 1 Bowl, 377.0ms
Speed: 8.0ms preprocess, 377.0ms inference, 1.5ms postprocess per image at shape (1, 3, 640, 640)
Detected 1 objects in /content/Deteksi-Bowl-1/test/images/mangkok-44-_jpg.rf.fd35bb9cd1ec89f3150f561c9bc4d316.jpg

0: 640x640 (no detections), 344.5ms
Speed: 3.7ms preprocess, 344.5ms inference, 0.8ms postprocess per image at shape (1, 3, 640, 640)
Detected 1 objects in /content/Deteksi-Bowl-1/test/images/mangkok-3-_jpg.rf.d6f1716a5c7a9dbf745ed9e31356b9ed.jpg

0: 640x640 1 Bowl, 359.9ms
Speed: 4.3ms preprocess, 359.9ms inference, 1.5ms postprocess per image at shape (1, 3, 640, 640)
Detected 1 objects in /content/Deteksi-Bowl-1/test/images/mangkok-22-_jpg.rf.123fc6ea654ccf2156d3e76d3fa78be9.jpg

0: 640x640 1 Bowl, 407.3ms
Speed: 3.7ms preprocess, 407.3ms inference, 1.8ms postprocess per image at shape (1, 3, 640, 640)
Detected 1 objects in /content/Deteksi-Bowl-1/test/images/mangkok-73-_jpg.rf.45a17e77250ea9c60f10079865676bd7.jpg
Time taken for detection on 4 images: 1.68 seconds
```

```
# VISUALIZE
for img_path in test_paths[:4]:
    detected_image = detect_objects(img_path)
    if detected_image is None:
        continue
    # Display detected image
    plt.figure(figsize=(6, 6))
    plt.imshow(detected_image)
    plt.axis('off')
    plt.title(f'Detected Objects in {os.path.basename(img_path)}')
    plt.show()
```

YOLO Results



HOG-SVM Data Preperation

```
[ ] !pip install roboflow

    from roboflow import Roboflow
    rf = Roboflow(api_key="pifrxtTxM8ZR0EIkia1x")
    project = rf.workspace("machine-learning-9xamz").project("deteksi-bowl")
    version = project.version(1)
    dataset = version.download("yolov8")
```

```
# DATA PREPARATION
def parse_yolo_annotation(annotation_path, img_width, img_height):
    bboxes = []
    with open(annotation_path, 'r') as file:
        for line in file:
            parts = line.strip().split()
            class_id = int(parts[0])
            if class_id == 0:
                x_center, y_center, width, height = map(float, parts[1:])
                # Convert normalized coordinates to pixel values
                x_min = int((x_center - width / 2) * img_width)
                y_min = int((y_center - height / 2) * img_height)
                x_max = int((x_center + width / 2) * img_width)
                y_max = int((y_center + height / 2) * img_height)
                bboxes.append((x_min, y_min, x_max, y_max))
    return bboxes
```

```
repare dataset function with limit
 prepare_dataset(image_dir, annotation_dir, limit=None):
 images = []
 labels = []
 count = 0

 for annotation_path in glob.glob(os.path.join(annotation_dir, "*.txt")):
     if limit is not None and count >= limit:
         break

     image_path = os.path.join(image_dir, os.path.basename(annotation_path).replace('.txt', '.jpg'
     image = cv2.imread(image_path)

     if image is None:
         continue

     img_height, img_width = image.shape[:2]
     bboxes = parse_yolo_annotation(annotation_path, img_width, img_height)

     # Resizing images and normalizing pixel values
     for (x_min, y_min, x_max, y_max) in bboxes:
         cropped_img = image[y_min:y_max, x_min:x_max]
         resized_img = cv2.resize(cropped_img, (64, 128))
         normalized_img = resized_img.astype(np.float32) / 255.0  # Normalizing pixel values
         images.append(normalized_img)
         labels.append(1)

     # Negative sample
     neg_sample = cv2.resize(image[0:128, 0:64], (64, 128))
     normalized_neg_sample = neg_sample.astype(np.float32) / 255.0  # Normalizing pixel values
     images.append(normalized_neg_sample)
     labels.append(0)

     count += 1

 return images, labels
```

HOG-SVM Model Implementation

```
[ ]  # Directories to access Dataset from Roboflow
     train_image_dir = os.path.join(dataset.location, "train", "images")
     train_annotation_dir = os.path.join(dataset.location, "train", "labels")
     test_image_dir = os.path.join(dataset.location, "valid", "images")
     test_annotation_dir = os.path.join(dataset.location, "valid", "labels")
     edge_case_image_dir = '/content/download.jpg'
```

```
# Extract HOG features
hog = cv2.HOGDescriptor()

def extract_hog_features(images):
    images_uint8 = [(image * 255).astype(np.uint8) for image in images]
    return [hog.compute(image).flatten() for image in images_uint8]

train_features = extract_hog_features(train_images)
val_features = extract_hog_features(val_images)
test_features = extract_hog_features(test_images)

# Model Building Using HOG-SVM

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'C': [0.1, 1, 10, 100],
    'max_iter': [1000, 2000, 3000]
}
```

HOG-SVM Training the Model

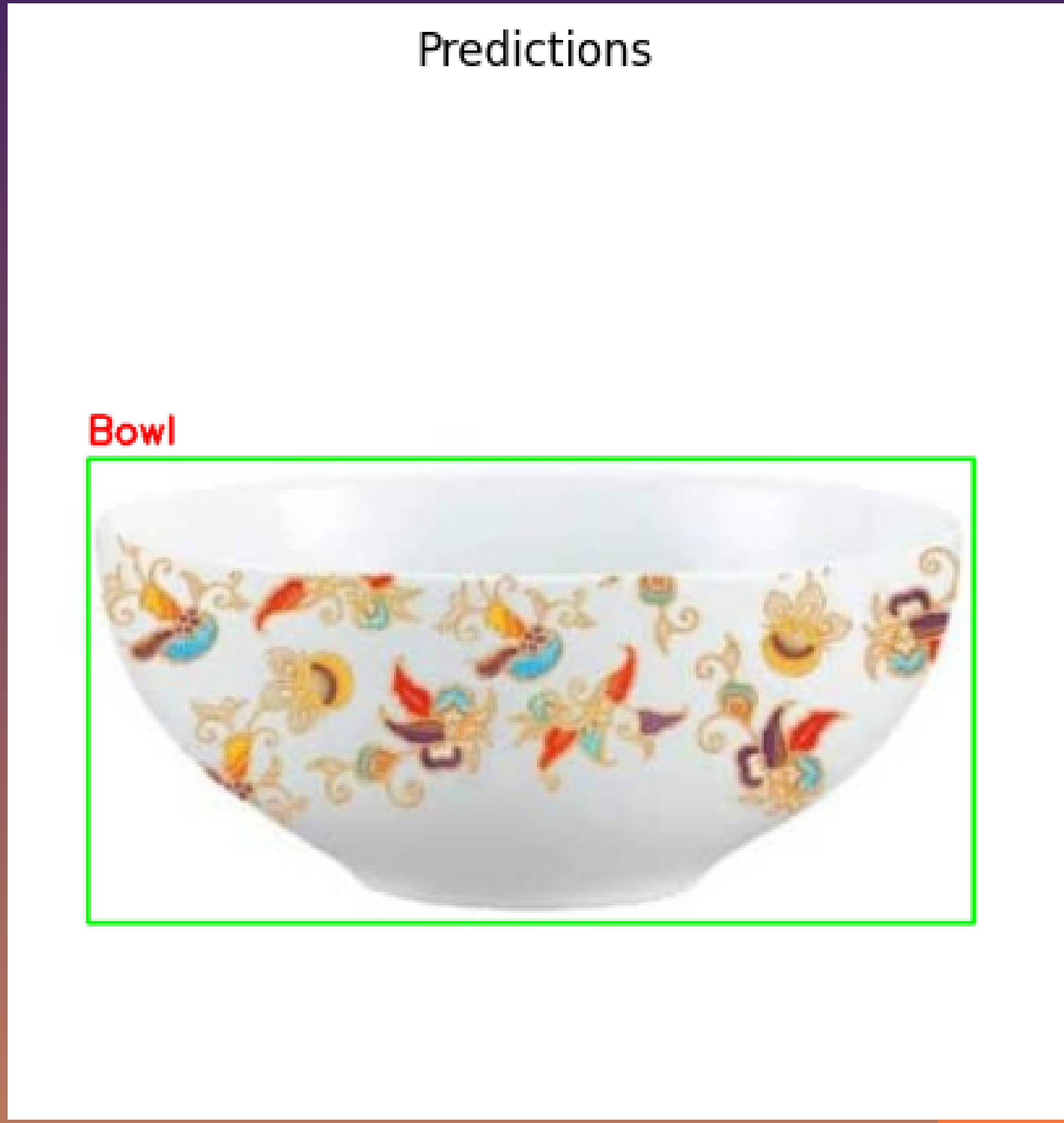
```python
# Load training and testing datasets
train_images, train_labels = prepare_dataset(train_image_dir, train_annotation_dir)
test_images, test_labels = prepare_dataset(test_image_dir, test_annotation_dir, limit=5)

# Split the training dataset into training and validation datasets
train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels, test_size=0.2, random_state=42)

# Extract HOG features
hog = cv2.HOGDescriptor()

def extract_hog_features(images):
    images_uint8 = [(image * 255).astype(np.uint8) for image in images]
    return [hog.compute(image).flatten() for image in images_uint8]

train_features = extract_hog_features(train_images)
val_features = extract_hog_features(val_images)
test_features = extract_hog_features(test_images)
```

HOG-SVM Evaluation and Testing

```python
import time

grid = GridSearchCV(LinearSVC(), param_grid, cv=5)
start_time = time.time()
grid.fit(train_features, train_labels)
training_time = time.time() - start_time
print("Best parameters found: ", grid.best_params_)
print("HOG-SVM model trained successfully in {:.2f} seconds.".format(training_time))

# Model evaluation on validation set
val_preds = grid.predict(val_features)

# Calculate performance metrics
val_accuracy = accuracy_score(val_labels, val_preds)
precision = precision_score(val_labels, val_preds)
recall = recall_score(val_labels, val_preds)

print("Validation Accuracy:", val_accuracy)
print("Precision:", precision)
print("Recall:", recall)

# Model evaluation on test set
test_preds = grid.predict(test_features)

# Calculate performance metrics for the test set
test_accuracy = accuracy_score(test_labels, test_preds)
precision = precision_score(test_labels, test_preds)
recall = recall_score(test_labels, test_preds)

print("Testing Accuracy:", test_accuracy)
print("Precision:", precision)
print("Recall:", recall)

# Speed Evaluation
start_time = time.time()
_ = grid.predict(test_features)  # Just for timing
detection_time = time.time() - start_time
print("Detection time for test set:", detection_time)
```

```python
#TESTING
def parse_yolo_annotation(annotation_path, img_width, img_height):
    bboxes = []
    with open(annotation_path, 'r') as file:
        for line in file:
            parts = line.strip().split()
            if len(parts) >= 5:  # Ensure there are enough parts
                # YOLO format: class_id x_center y_center width height
                class_id, x_center, y_center, width, height = map(float, parts)
                x_center, y_center, width, height = (
                    x_center * img_width,
                    y_center * img_height,
                    width * img_width,
                    height * img_height
                )
                x_min = int(x_center - width / 2)
                y_min = int(y_center - height / 2)
                x_max = int(x_center + width / 2)
                y_max = int(y_center + height / 2)
                bboxes.append((x_min, y_min, x_max, y_max))
    return bboxes

def visualize_predictions(image_dir, annotations_dir, predictions, limit=10):
    count = 0
    for annotation_path in glob.glob(os.path.join(annotations_dir, "*.txt")):
        image_path = os.path.join(image_dir, os.path.basename(annotation_path).replace('.txt', '.jpg'))
        image = cv2.imread(image_path)

        if image is None:
            print(f"Failed to load image: {image_path}")
            continue

        img_height, img_width = image.shape[:2]
        bboxes = parse_yolo_annotation(annotation_path, img_width, img_height)

        # Debug: Print bounding boxes
        print(f'Bounding Boxes: {bboxes}')
```

HOG-SVM Result



EVALUATION

Histogram of Oriented Gradients with Support Vector Machine (HOG-SVM) and You Only Look Once (YOLO) both reached a high percentage of accuracy. Though both have no problems during the training process as well as the results that were provided.

COMPARISON

Histogram of Oriented Gradients with Support Vector Machine (HOG-SVM) and You Only Look Once (YOLO) runned well in google colab. The difference was HOG-SVM has its fastest training time compared to YOLO. Based on the programmers observation, YOLO trained the pre-trained dataset wherein it caused the slow progress of output using the algorithm as it involves fine-tuning weights across multiple layers. HOG-SVM requires fewer computational resources, as it focuses on detecting shapes and edges, it had gained the most accurate result due to the effect of training ten (10) epochs although the suggested minimum number of epochs was fifty (50).

Thank You!

GAGALANG
GALZOTE
CS4A