Midterm Project
Implementing Object Detection on a Dataset

Selection of Dataset and Algorithm

The Chess Board Detection Dataset is made to detect and locate chessboards in images. With 74 images, the dataset is divided into training, validation, and test sets, which helps with training models, tuning parameters, and testing performance. About 85% of the images (63) are used for training, 8% (6 images) for validation, and 7% (5 images) for testing.

```
from roboflow import Roboflow
rf = Roboflow(api_key="9wNF5Bb9cpKIvNwqpVYu")
Project = rf.workspace("tutorial-knlj8").project("chess-board-detection-uuppk")
version = project.version(4)
dataset = version.download("yolov11")
```

The dataset was prepared using Roboflow to download a specific version of the chessboard detection dataset, formatted for compatibility with YOLO. This ensured that the images and annotations aligned with the YOLO model's requirements.

```
from ultralytics import YOLO
# Load a pretrained model from the Ultralytics HUB
model = YOLO("/content/yolo11m.pt")
```

Ultralytics' YOLO library was used to load a pre-trained YOLO model and initiate training. This pre-trained model served as a solid foundation, utilizing prior knowledge from general object detection tasks to improve the accuracy of detecting chessboards.

Implementation

Data Preparation

The dataset undergoes several preprocessing steps to ensure consistency and quality for effective model training. First, auto-orientation is applied to each image, standardizing their alignment for a uniform perspective. To maintain the relevance of the data, a filtering process is implemented, requiring that at least 85% of each image contains annotations. This step helps ensure that only adequately annotated images are included in the dataset.

To enhance model performance, various augmentations are applied to the training images. For each image, three variations are created, which include horizontal flips to introduce mirrored perspectives and slight rotations ranging from -2° to +2° to cover different viewing angles. Additionally, brightness adjustments are made, allowing variations from -16% to +16% to simulate various lighting conditions.

Furthermore, the Albumentations library is employed to introduce additional augmentations, which include:

- **Blur**: A slight blur effect applied with a probability of 0.01, using a blur limit ranging from 3 to 7 pixels, helping to reduce noise and enhance generalization.
- **MedianBlur**: Another blurring technique applied with the same probability and limit can effectively remove outliers and noise in the images.
- **ToGray**: This augmentation converts images to grayscale with a probability of 0.01, using a weighted average method to maintain a three-channel output. This can help the model learn to detect features in varying color spaces.
- **CLAHE (Contrast Limited Adaptive Histogram Equalization)**: Applied with a probability of 0.01, this technique enhances contrast in images by applying local contrast adjustments, improving the visibility of features in different lighting conditions. The parameters include a clip limit ranging from 1 to 4.0 and a tile grid size of 8x8, allowing for fine control over the enhancement process.

To further improve robustness, random noise is added to up to 3% of pixels, mimicking real-world variations in image quality. Lastly, the brightness of bounding boxes is adjusted by -4% to +4%, helping the model learn more consistent boundary features.

Model Building

In the model-building phase, the optimization process is a critical component that directly influences the training efficiency and performance of the model. Initially, an automatic optimizer selection is performed, indicated by the message:

```
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and
determining best 'optimizer', 'lr0' and 'momentum' automatically...
```

This step signifies that predefined values for the learning rate (lr0) and momentum are disregarded. Instead, the system will dynamically assess and select the most suitable optimizer and optimal parameters based on the dataset and task.

The outcome of this process is the selection of the AdamW optimizer, which is known for its effectiveness in training deep learning models. The parameters determined for this optimizer are:

- Learning Rate (lr): 0.000625
- -Momentum: 0.9

The parameter groups configured for the optimizer include:

- 106 weights with no weight decay (decay=0.0)
- 113 weights with a weight decay of 0.0005
- 112 biases with no weight decay (decay=0.0)

AdamW allows for better handling of weight decay through decoupling the weight decay from the gradient updates, which can enhance generalization and reduce overfitting. By employing this adaptive learning strategy, the model can adjust the learning rate dynamically based on the gradients, leading to more stable and efficient convergence during training.

Training the Model

```
# Train the modelresults =
model.train(data="/content/Chess-Board-Detection-4/data.yaml", epochs=100,
imgsz=640)
```

This line calls the train method on the model object, an instance of the YOLO class. The method trains the model using the provided configuration and dataset.

Parameters:

data="/content/Chess-Board-Detection-4/data.yaml"

This parameter specifies the path to the data.yaml file contains information about the dataset, including the paths to the training, validation, test sets, class names, and other configuration details. It is crucial for guiding the training process, as the model needs to know where to find the images and their corresponding annotations.

epochs=100

This parameter sets the number of training epochs to 100. An epoch is one complete pass through the entire training dataset. Setting a higher number of epochs allows the model more opportunities to learn from the data, although it also increases the training time. Depending on the model's performance, early stopping can terminate training if no improvement is observed.

imgsz=640

This parameter defines the image size to which all input images will be resized during training. Setting imgsz=640 means images will be resized to a width and height of 640 pixels. Resizing images ensures consistent input dimensions, which is important for deep-learning models and can help speed up training.

Testing and Evaluation

```
from ultralytics import YOLO
```

This line imports the YOLO class from the Ultralytics library, which contains methods for handling YOLO object detection models.

```
model = YOLO("/content/runs/detect/train/weights/best.pt")  # load a custom
model
```

This line loads a custom-trained YOLO model from the specified path (/content/runs/detect/train/weights/best.pt). This model file contains the weights and configuration necessary for the trained model.

```
# Validate the model
metrics = model.val()  # no arguments needed, dataset and settings remembered
```

The val() method is called on the model object to validate its performance on the validation dataset. The method automatically uses the dataset and settings that were defined during training.

Output:

```
YOLO11m summary (fused): 303 layers, 20,039,284 parameters, 0 gradients, 67.7
GFLOPs
```

This summary provides information about the architecture of the YOLO model, including the number of layers (303), the total number of parameters (20,039,284), the gradients (0 indicates that the model is not currently being trained), and the computational complexity in GFLOPs (Giga Floating Point Operations per second).

```
val: Scanning /content/Chess-Board-Detection-4/valid/labels.cache... 6 images, 0
backgrounds, 0 corrupt: 100%|          | 6/6 [00:00<?, ?it/s]
```

This line indicates that the validation process scans the specified validation dataset comprising 6 images. It confirms that there are no background or corrupt images.

```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
```

This section summarizes the validation results, showing metrics for each class detected in the dataset:

- ***Images:*** The number of images containing the specific class.
- ***Instances:*** The total number of instances of that class detected.

- ***Box(P):*** Precision for the bounding box predictions.
- ***R:*** Recall, indicating the ability to find all relevant instances.
- ***mAP50:*** Mean Average Precision at an IoU threshold of 0.50.
- ***mAP50-95:*** Mean Average Precision averaged over IoU thresholds from 0.50 to 0.95.

```
Speed: 0.3ms preprocess, 41.8ms inference, 0.0ms loss, 1.1ms postprocess per
image
```

This indicates the time taken for various stages of the processing pipeline per image:

- ***Preprocessing:*** 0.3 ms
- ***Inference:*** 41.8 ms
- ***Loss calculation:*** 0.0 ms (not applicable during validation)
- ***Post Processing:*** 1.1 ms

```python
import numpy as np

# Access the confusion matrix
confusion_matrix = metrics.confusion_matrix.matrix

# Get true positive (TP), false positive (FP), and false negative (FN) from the
matrix
TP = np.diag(confusion_matrix)  # True Positives are diagonal elements
FP = np.sum(confusion_matrix, axis=0) - TP  # False Positives per class
FN = np.sum(confusion_matrix, axis=1) - TP  # False Negatives per class

# Calculate precision and recall for each class
precision = TP / (TP + FP)  # Precision per class
recall = TP / (TP + FN)  # Recall per class

# Handle potential division by zero errors
precision = np.nan_to_num(precision, nan=0.0, posinf=0.0, neginf=0.0)
recall = np.nan_to_num(recall, nan=0.0, posinf=0.0, neginf=0.0)

# Calculate overall precision and recall (macro average)
overall_precision = np.mean(precision)
overall_recall = np.mean(recall)

# Calculate accuracy
overall_accuracy = np.sum(TP) / np.sum(confusion_matrix)
# Print the results
print(f"Overall Precision: {overall_precision:.2f}")
print(f"Overall Recall: {overall_recall:.2f}")
print(f"Overall Accuracy: {overall_accuracy:.2f}")
```

To analyze the model's predictions, the code calculates three key metrics: True Positives (TP), False Positives (FP), and False Negatives (FN). True Positives are identified along the diagonal of the confusion matrix, representing instances where the model correctly predicted the class. False Positives are determined by summing the predicted instances for each class and subtracting the True Positives, indicating the number of incorrect positive predictions. False Negatives are computed by summing the actual instances for each class and subtracting the True Positives, showing the missed positive cases.

With these metrics in hand, the code calculates precision and recall for each class. Precision reflects the accuracy of the model's positive predictions, while recall measures the model's ability to identify all relevant instances. To address potential division by zero issues, the code ensures that any undefined values are replaced with zero, allowing for a robust evaluation.

Next, the overall precision and recall are computed by taking the mean of the class-specific values. This macro-average approach provides a holistic view of the model's performance across all classes. Additionally, the code calculates the overall accuracy, representing the proportion of correctly classified instances out of the total predictions made by the model.

Performance Metrics

Table 1. Performance Metrics for YOLO

Class	Images	Instances	Box(P	R	mAP50	mAP50-95
all	6	118	0.954	0.95	0.985	0.645
black-camel	5	10	0.976	0.9	0.986	0.655
black-elephant	5	9	0.959	1	0.995	0.61
black-horse	5	10	1	0.783	0.995	0.576
black-king	5	5	0.936	1	0.995	0.666
black-pawn	5	34	0.937	0.971	0,953	0.555
black-queen	4	4	1	0.968	0.995	0.706
white-camel	4	8	1	0.949	0.995	0.677
white-elephant	3	6	0.943	1	0.995	0.564
white-horse	4	8	1	0.983	0.995	0.748
white-king	3	3	1	0.896	0.995	0.679
white-pawn	3	18	0.88	0.994	0.927	0.604
white-queen	3	3	0.815	1	0.995	0.703

For all classes combined, the model shows a precision of 0.954 and a recall of 0.95, indicating high accuracy and detection ability. The mean average precision (mAP) at IoU 0.5 is 0.985, demonstrating excellent performance at this threshold, while the mAP from 0.5 to 0.95 is 0.645, suggesting that the model's performance varies with stricter IoU requirements.

The class-specific performance results reveal insightful metrics regarding the model's detection capabilities for each type of chess piece. Starting with the black-camel, the model demonstrated high precision at 0.976 and a commendable recall of 0.9. This performance indicates that the model reliably detected this class, achieving a strong balance between accurate predictions and capturing the majority of instances present in the validation set.

In contrast, the black-elephant exhibited a perfect recall of 1.0, signifying that the model successfully identified all instances of this class. However, its precision was slightly lower at 0.959, suggesting that while the model was thorough in its detections, it also encountered some false positives.

The black-horse showed an impressive precision score of 1.0, indicating that every prediction made for this class was correct. Nonetheless, the model's recall was lower at 0.783, implying that it missed some instances of the black-horse during detection.

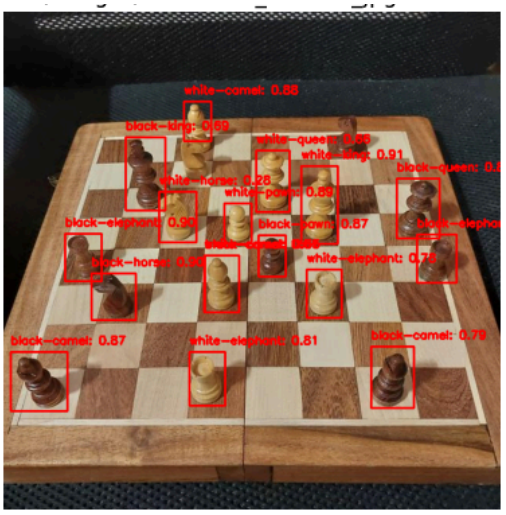
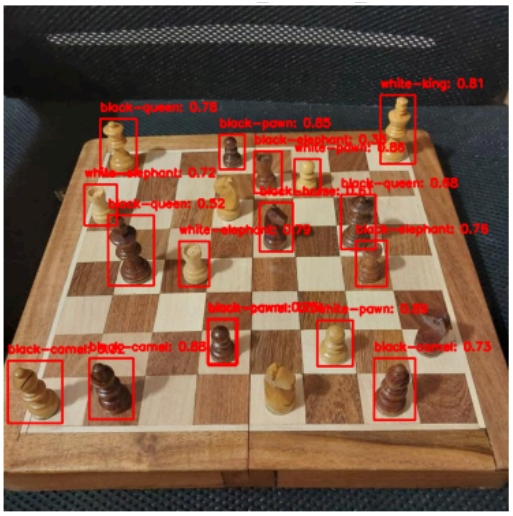
The metrics reflect a well-balanced performance regarding precision and recall for the remaining classes, such as the black-pawn and white-pawn. This balance indicates effective detection capabilities, as the model managed to identify the pieces without generating an excessive number of false predictions.

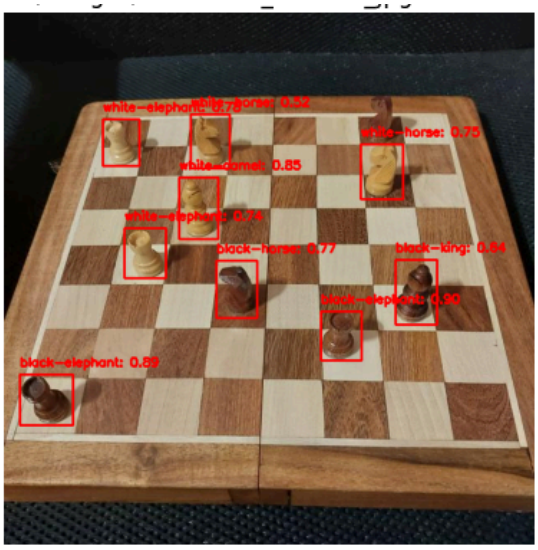
Table 2. Overall Metrics for YOLO

Overall Precision	0.88
Overall Recall	0.85
Overall Accuracy	0.92

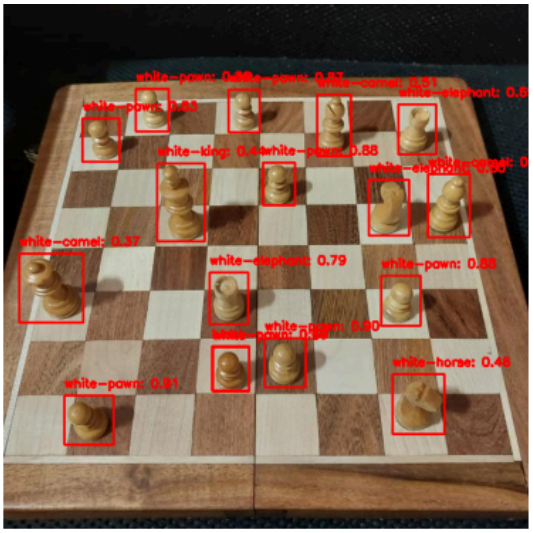
The table reveals that the model achieves an overall precision of 0.88, indicating that 88% of the instances predicted as positive were indeed correct. The recall score of 0.85 suggests that the model successfully identified 85% of the actual positive instances, reflecting its capability to detect relevant cases. With an overall accuracy of 0.92, the model demonstrates a high classification performance, successfully identifying 92% of all instances.

Testing the model using the testing set

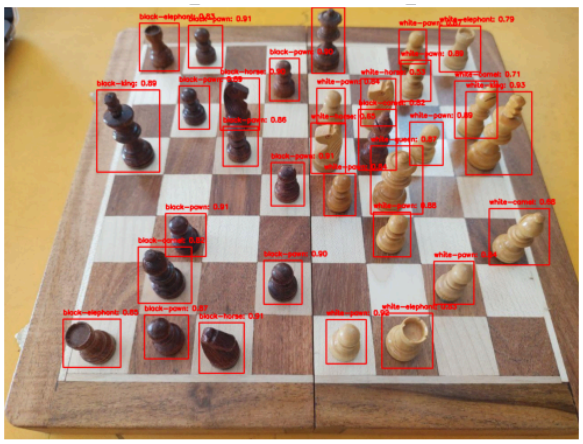
Processed Image	Inference Time
	40.3 milliseconds
	36.4 milliseconds



36.4 milliseconds



36.4 milliseconds



30.1 milliseconds

Comparison

HOG-SVM

Table 3. Performance Metrics of Training Class for HOG-SVM

Class	Precision	Recall	F1-Score	Support
black-camel	1.00	1.00	1.00	51
black-elephant	1.00	1.00	1.00	51
black-horse	1.00	1.00	1.00	51
black-king	1.00	1.00	1.00	51
black-pawn	1.00	1.00	1.00	39
black-queen	1.00	1.00	1.00	51
white-camel	1.00	1.00	1.00	54
white-elephant	1.00	1.00	1.00	51
white-horse	1.00	1.00	1.00	51
white-king	1.00	1.00	1.00	57
white-pawn	1.00	1.00	1.00	42
white-queen	1.00	1.00	1.00	54
micro avg	1.00	1.00	1.00	603
macro avg	1.00	1.00	1.00	603
weighted avg	1.00	1.00	1.00	603
samples avg	1.00	1.00	1.00	603

Training Accuracy: 1.0

The model achieved perfect performance across all metrics—precision, recall, and F1-score—for each class of chess pieces, including all black and white camels, elephants, horses, kings, and queens. This means that every instance of each piece was correctly identified, with no false positives or missed cases. Precision is 1.00 across the board, indicating that every prediction was accurate, and recall is also 1.00, meaning all instances were captured.

The F1-score mirrors this flawless performance, confirming the model's consistent handling of all classes. Although support varies from 39 (black-pawn) to 57 (white-king), accuracy remains uniformly perfect. Micro, macro, and weighted averages all stand at 1.00, reinforcing the model's balance and effectiveness.

While these results suggest exceptional training performance, there may be overfitting. Testing on validation or test data will determine if the model generalizes well beyond the training data.

Table 4. Performance Metrics of Validation Class for HOG-SVM

Class	Precision	Recall	F1-Score	Support
black-camel	1.00	0.80	0.89	5
black-elephant	0.75	0.60	0.67	5
black-horse	1.00	0.80	0.89	5
black-king	0.75	0.60	0.67	5
black-pawn	1.00	0.40	0.57	5
black-queen	0.75	0.75	0.75	4
white-camel	1.00	0.50	0.67	4
white-elephant	0.50	0.67	0.57	3
white-horse	1.00	0.25	0.40	4
white-king	0.67	0.67	0.67	3
white-pawn	1.00	0.67	0.80	3
white-queen	0.67	0.67	0.67	3
micro avg	0.81	0.61	0.70	49
macro avg	0.84	0.61	0.68	49
weighted avg	0.86	0.61	0.69	49
samples avg	0.69	0.54	0.58	49

Validation Accuracy: 0.16666666666666666

For instance, the model achieves high precision (1.00) but only moderate recall (0.80) for black-camel and black-horse pieces, yielding strong F1 Scores of 0.89. However, classes like black-elephant and black-king have lower precision and recall (0.75 and 0.60, respectively), resulting in F1 Scores of 0.67. Similarly, black-pawn and white-horse show high precision (1.00) but low recall (0.40 and 0.25, respectively), indicating missed detections.

The micro-average (precision 0.81, recall 0.61) suggests moderate accuracy but missed instances. The macro-average F1-score of 0.68 highlights an imbalance in class performance. With an overall validation accuracy of 16.7%, the model identifies all classes correctly in only a few cases, indicating generalization challenges. In summary, the model shows good precision in some areas but struggles with recall, suggesting a need for further adjustments to improve its validation performance.

Table 5. Performance Metrics of Test Class for HOG-SVM

Class	Precision	Recall	F1-Score	Support
black-camel	1.00	0.75	0.86	4
black-elephant	1.00	0.75	0.86	4
black-horse	1.00	0.75	0.86	4
black-king	0.67	1.00	0.80	2
black-pawn	1.00	1.00	1.00	3
black-queen	1.00	1.00	1.00	3
white-camel	1.00	1.00	0.75	5
white-elephant	1.00	1.00	0.75	5
white-horse	1.00	1.00	0.75	5
white-king	1.00	1.00	1.00	4
white-pawn	1.00	0.75	0.86	4
white-queen	1.00	0.75	0.86	4
micro avg	0.97	0.77	0.86	47
macro avg	0.97	0.80	0.86	47
weighted avg	0.99	0.77	0.85	47

samples avg	0.78	0.65	0.68	47

Test Accuracy: 0.2

Classes like black-camel, black-elephant, and black-horse display a precision of 1.00 but a recall of 0.75, leading to an F1-score of 0.86. This indicates the model's strong capability to correctly identify these pieces when detected, though it occasionally misses instances. For classes such as black-king and black-queen, recall reaches 1.00 with precision between 0.67 and 1.00, showing that these pieces are well recognized in an image.

The micro-average F1-score is 0.86, indicating moderate detection accuracy across all pieces. The macro-average F1-score also aligns at 0.86, but with a higher variance across classes due to differing recall rates. The weighted average precision of 0.99 reflects a strong confidence in correctly classified detections, although test accuracy sits lower at 20%.

In summary, while the model demonstrates solid precision in recognizing specific chess pieces, achieving consistent recall across all classes remains challenging, affecting the overall detection reliability.

Table 6. Overall Metrics for HOG-SVM

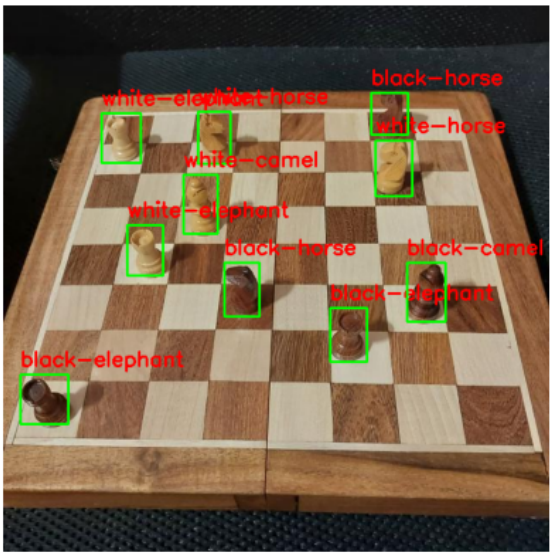
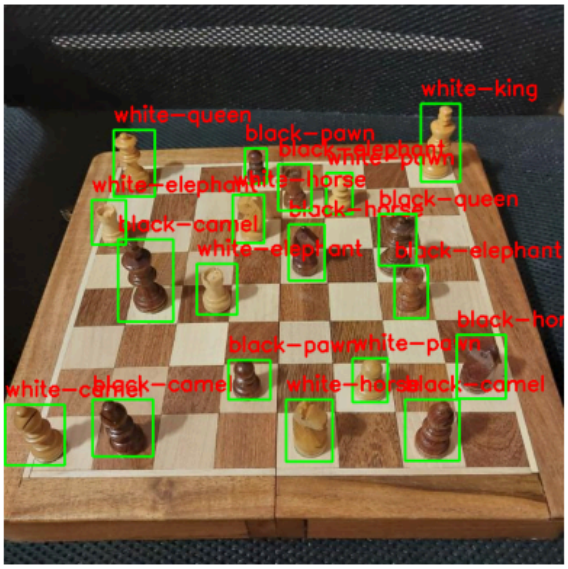
Validation Accuracy	0.16666666666666666
Validation Precision	0.8402777777777777
Validation Recall	0.6138888888888889
Test Accuracy	0.2
Test Precision	0.9722222222222222
Test Recall	0.7958333333333334
Detection time for test set	0.005112171173095703

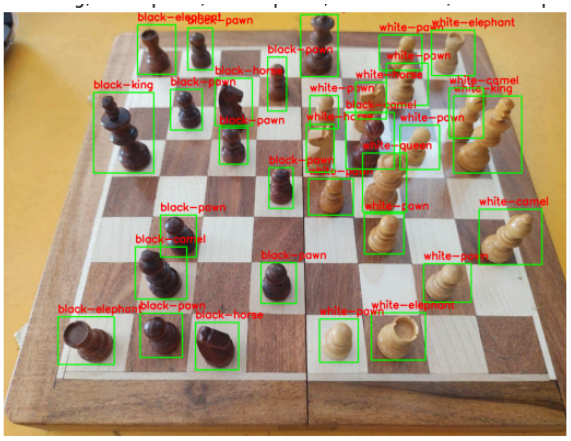
In validation, the model’s accuracy is low at 0.17, indicating it correctly classified only 16.7% of the samples. Precision, however, is considerably higher at 0.84, suggesting that it is mostly accurate when the model makes a positive prediction. Recall, or the ability to capture all true instances, stands at 0.61, meaning some cases were missed.

On the test set, accuracy improves to 0.20, and precision is high at 0.97, reflecting a strong confidence in its predictions. Test recall is 0.80, indicating better coverage of true

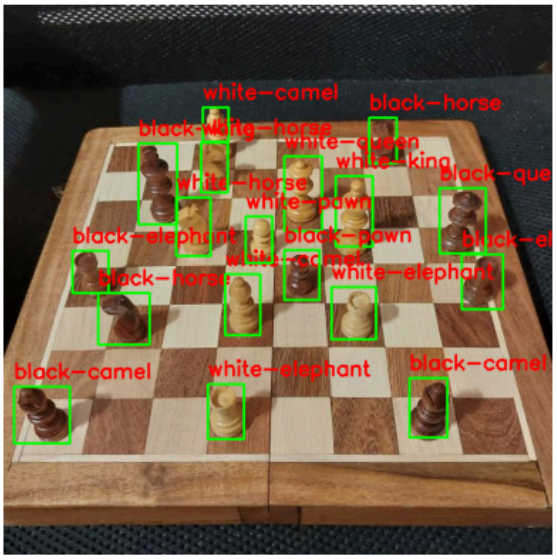
instances than the validation set. This suggests the model captures most of the true cases in the test set but still misses some. Detection time per image in the test set is efficient, averaging around 0.005 seconds.

These results reveal a model that is precise in its predictions but struggles with recall, particularly in the validation set. This indicates potential challenges in generalizing to unseen data.

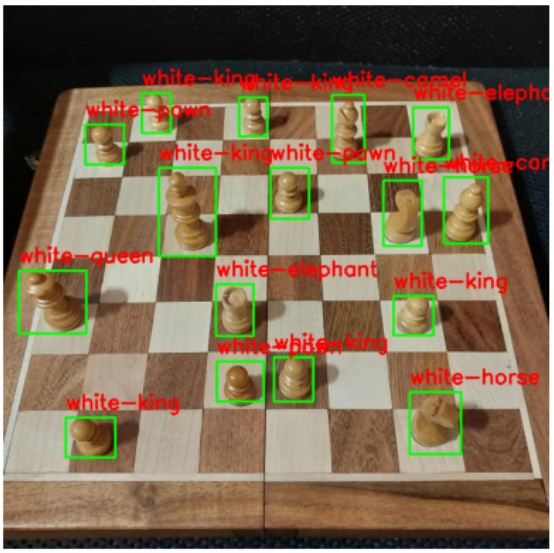
Processed Image	Inference Time
	0.0154 seconds
	0.0099 seconds



0.0190 seconds



0.0353 seconds



0.0091 seconds

MobileNet SSD

Table 7. Mean Average Precision for SSD

Class	mAP @ 0.5:0.95
black-camel	38.71%
black-elephant	9.17%
black-horse	44.83%
black-king	40.00%
black-pawn	10.00%
black-queen	20.00%
white-camel	15.36%
white-elephant	39.50%
white-horse	36.83%
white-king	7.78%
white-pawn	3.00%
white-queen	12.50%
overall	23.14%

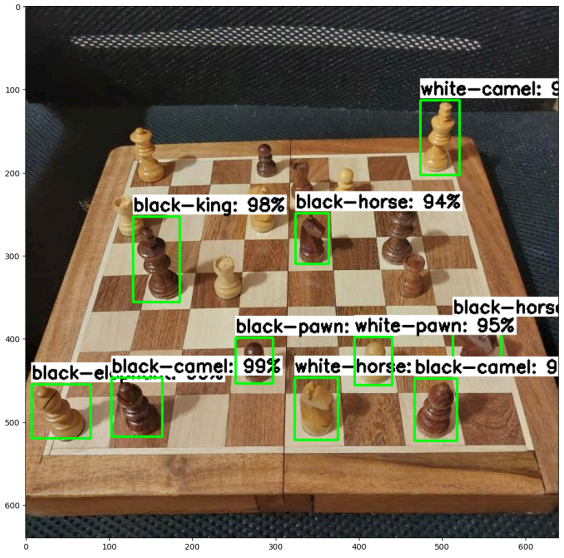
The highest detection scores are observed for pieces like the black-horse (44.83%) and black-camel (38.71%), reflecting the model's relative success in recognizing these classes across a range of IoU thresholds. Other pieces, like the white-elephant (39.50%) and black-king (40.00%), also demonstrate strong detection rates, indicating that the model has learned specific features that aid in detecting these objects more accurately.

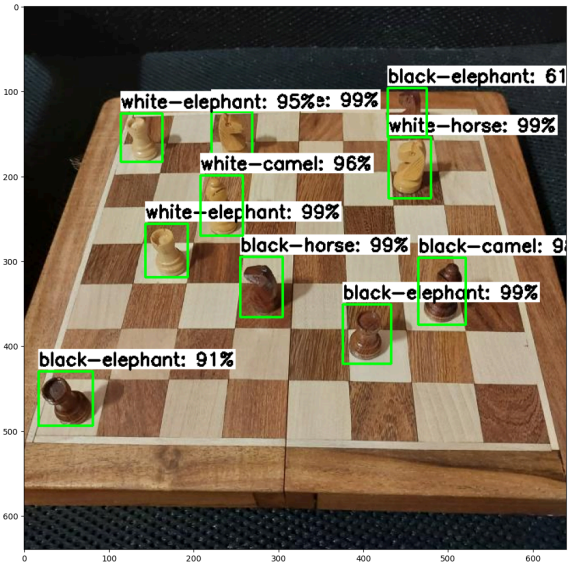
In contrast, certain pieces show notably low detection rates. The white-pawn (3.00%) and white-king (7.78%) score significantly lower, highlighting difficulty in consistent detection for these pieces. This may be due to subtle features or high similarity between pawns and other pieces, making them harder to identify.

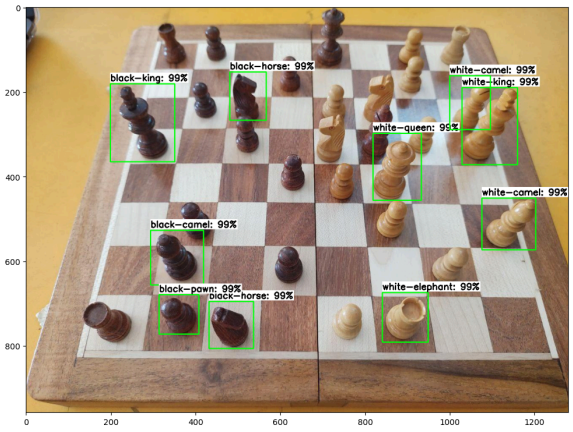
The average mAP @ 0.5:0.95 for all classes is 23.14%, suggesting moderate performance under the challenging mAP @ 0.5:0.95 metric. This metric’s use of varying IoU thresholds reveals that while some classes perform well, overall consistency across all

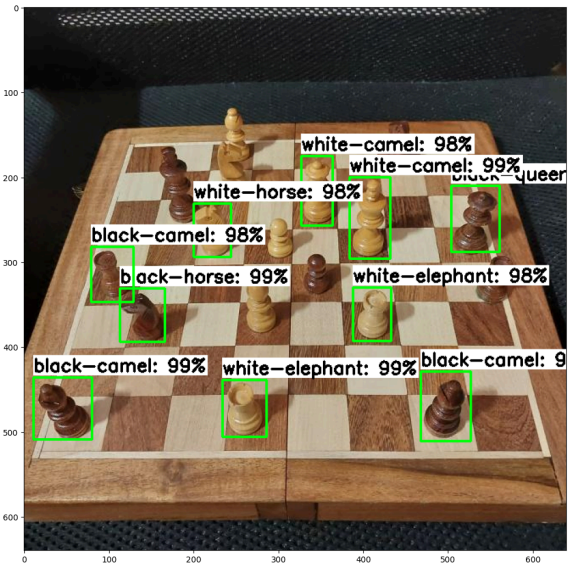
classes could be improved. Higher scores in specific classes imply that the model has learned meaningful representations for certain pieces. In comparison, the lower scores in others indicate room for refinement, possibly through additional data or targeted training for underperforming classes.

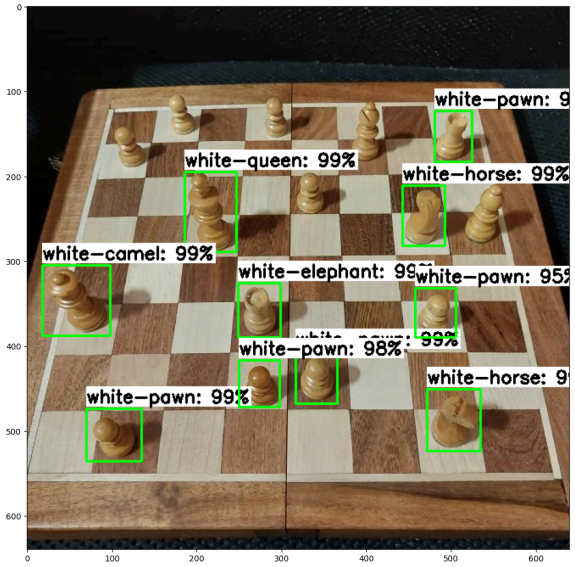
In summary, while the model is capable of effectively detecting certain chess pieces, an overall mAP of 23.14% suggests that further optimization is needed to enhance consistency and accuracy across all chess piece classes, especially those with lower average precision scores.











Inference Time for 5 images: 0.0011 seconds

Average Inference Time per image: 0.0002 seconds

Overall Analysis

YOLO Model Analysis

Overall Metrics:

- **Precision:** 0.88
- **Recall:** 0.85
- **Accuracy:** 0.92
- **mAP50:** High precision (≥0.95) in detecting several classes.
- **mAP50-95**: A lower average mAP50-95 (0.645) suggests potential difficulty detecting objects at different IoU thresholds.

Class-Specific Observations:

High-performing classes include black-queen, white-horse, and white-king, with mAP50-95 values above 0.70. Lower-performing classes include black-horse and white-pawn, where mAP50-95 is between 0.55 and 0.6, indicating some variance in detection performance for these classes.

HOG-SVM Model Analysis

Training Metrics:

It achieved perfect scores for Precision, Recall, and F1-Score (all 1.0), indicating strong model performance on the training set. This might suggest potential overfitting, as these metrics often drop on validation and test sets.

Validation and Test Metrics:

- **Validation Accuracy:** 0.167, indicating a significant drop from training, which aligns with potential overfitting.
- **Test Accuracy:** 0.2, with a noticeable disparity in Precision (0.97) and Recall (0.80), suggesting that the model performs well when it detects an object but struggles to detect all instances.

Class-Specific Observations:

- **Test Set:** Classes like black-king and black-elephant have lower F1-Scores, showing inconsistent detection on less common instances.
- **Performance on less frequent classes:** Classes with fewer instances (e.g., white-queen, white-pawn) saw reduced recall, likely impacting overall model generalizability.

MobileNet SSD Analysis

Overall Performance:

mAP @ 0.5:0.95: 23.14%, significantly lower than YOLO's, suggesting MobileNet SSD struggles with consistent detections across different IoU thresholds.

Class-Specific Observations:

High-performing classes: black-horse (44.83%) and black-king (40.00%) perform relatively better.

Low-performing classes: white-pawn (3%) and white-king (7.78%) indicate room for improvement, possibly due to less training data for these classes.

Summary

- YOLO outperforms HOG-SVM and MobileNet SSD in object detection accuracy, mAP, and detection consistency across classes.
- HOG-SVM performs excellently on the training set but shows a marked drop in validation and test performance, possibly due to overfitting.
- MobileNet SSD has relatively low mAP across classes, suggesting it may require additional tuning or more training data to match YOLO's performance.

Recommendations

- **Data Augmentation:** Enhancing data variability could help HOG-SVM generalize better, reducing overfitting.
- **Model Tuning:** For MobileNet SSD, experimenting with learning rates, batch sizes, and anchor box adjustments may help boost mAP.
- **Class Balance:** Some classes (e.g., white-pawn and white-king) show lower performance across models, which might be improved by balancing instances across all classes.

Table 8. Summary Table of Each Model

Model	Metric	Overall	Black Camel	Black Elephant	Black Horse	Black King	Black Pawn	Black Queen
YOLO	Precision	0.88	0.976	0.959	1.000	0.936	0.937	1.000
	Recall	0.85	0.900	1.000	0.783	1.000	0.971	0.968
	mAP50	0.985	0.985	0.986	0.995	0.995	0.953	0.995
	mAP50-95	0.645	0.645	0.655	0.576	0.666	0.555	0.706
HOG-SVM	Precision (Test)	0.97	1.000	1.000	1.000	0.667	1.000	1.000
	Recall (Test)	0.80	0.750	0.750	0.750	1.000	1.000	1.000
MobileNet SSD	mAP @ 0.5:0.95	0.2314	0.3871	0.0917	0.4483	0.4000	0.1000	0.2000

Table 8. (continued)

Model	Metric	White Camel	White Elephant	White Horse	White King	White Pawn	White Queen
YOLO	Precision	1.000	0.943	1.000	1.000	0.880	0.815
	Recall	0.949	1.000	0.983	0.896	0.994	1.000
	mAP50	0.995	0.995	0.995	0.995	0.927	0.995
	mAP50-95	0.677	0.564	0.748	0.679	0.604	0.703
HOG-SVM	Precision (Test)	1.000	1.000	1.000	1.000	1.000	1.000
	Recall (Test)	1.000	1.000	1.000	1.000	0.750	0.750
MobileNet SSD	mAP @ 0.5:0.95	0.1536	0.3950	0.3683	0.0778	0.0300	0.1250

YOLO shows strong performance across all metrics, with consistently high precision and recall values. The mAP50-95 metric highlights that some classes, like black-pawn and black-horse, perform slightly lower in multi-threshold detection accuracy, suggesting these objects might be more challenging for YOLO.

HOG-SVM performs well in terms of precision in the test set (0.97), with a lower recall (0.80). This disparity suggests HOG-SVM is very accurate in detecting objects when it does find them but misses some detections, as seen in classes like black-king and white-pawn.

MobileNet SSD has the lowest mAP50-95 performance, averaging 23.14%. Notably, black-horse (44.83%) and black-camel (38.71%) performed the best, but many other classes, like white-king and white-pawn, were detected poorly.

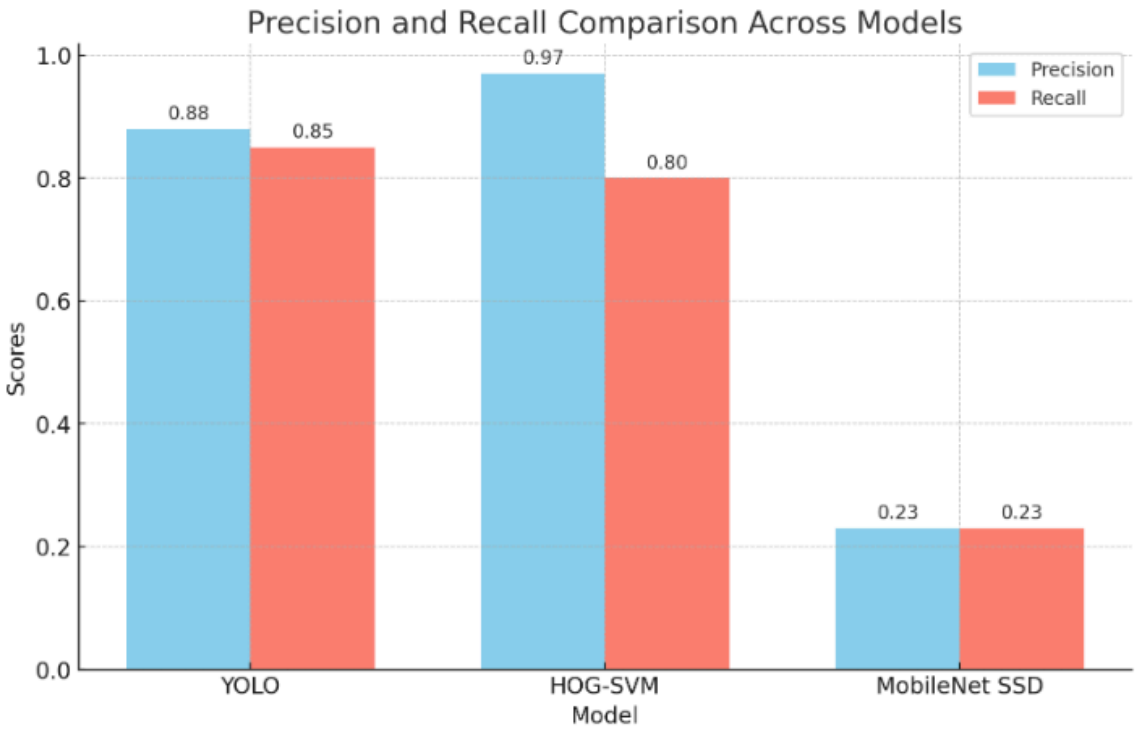


Figure 1. Precision and Recall Comparison Across Models

This bar chart highlights the performance difference in precision and recall across the three models. HOG-SVM achieved the highest precision (0.97) but had a lower recall (0.80), indicating it detects objects accurately but misses some. YOLO performed consistently in both metrics with balanced precision and recall (0.88 and 0.85, respectively), making it a reliable model for detection. MobileNet SSD has low precision and recall (0.23), suggesting it struggles with accurate and consistent detections.

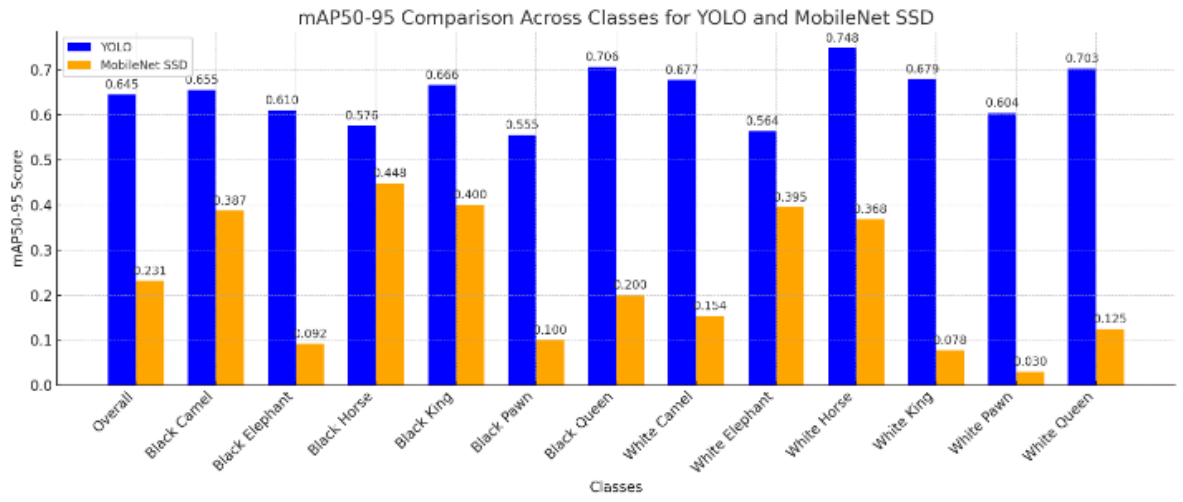


Figure 2. Mean Average Precision Comparison Across Classes for YOLO and SSD

This chart shows mAP50-95 scores across different chess piece classes for YOLO and MobileNet SSD. YOLO generally performed better across all classes, with notable high scores for Black Queen, White Horse, and White Queen, indicating consistent multi-threshold accuracy. In contrast, MobileNet SSD had lower scores overall, with some variability; for instance, Black Horse and Black Camel classes had comparatively better scores, but several classes, such as White King and White Pawn, performed poorly.

These visualizations and the summary table demonstrate that YOLO is generally more robust across various metrics and classes. At the same time, HOG-SVM and MobileNet SSD show strengths in specific but limited areas.

Table 9. Inference Speed of Each Model

Model	Inference Time per Image	Notes
YOLO	Average: 35.92 ms	5 inferences: 40.3 ms, 36.4 ms (x3), 30.1 ms
HOG-SVM	Average: 0.01774 s (17.74 ms)	5 inferences: 15.4 ms, 9.9 ms, 19.0 ms, 35.3 ms, 9.1 ms
SSD	0.0002 s (0.2 ms)	Per image; calculated from 5 images

From this table, YOLO and HOG-SVM take around 35 ms and 17.7 ms on average per image, respectively, with YOLO slightly faster. SSD, on the other hand, is extremely fast, taking just 0.2 ms per image on average, which makes it the quickest model but potentially at the cost of accuracy.

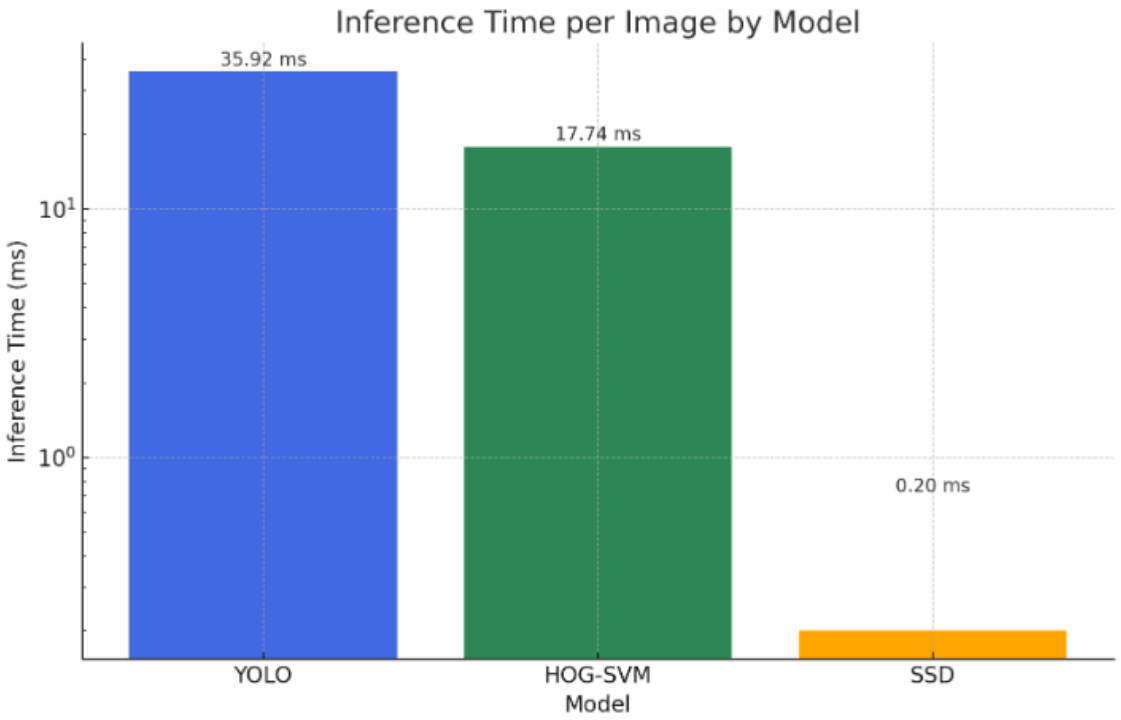


Figure 3. Inference Time per Image by Each Model

This bar chart shows the average inference time per image for each model, using a logarithmic scale to highlight differences across models. SSD is the fastest model, with an average inference time of just 0.2 milliseconds per image. This speed is achieved at the expense of some detection accuracy. HOG-SVM follows, with an average inference time of 17.74 milliseconds per image. Its speed is notably higher than SSD but lower than YOLO. YOLO has an average inference time of 35.92 milliseconds, making it the slowest among these models but with a balanced trade-off between speed and accuracy.

This graph emphasizes that while SSD is optimal for applications where speed is a priority, YOLO and HOG-SVM offer more balanced results suitable for tasks where precision and recall are also important factors.

Table 10. Summary Table of Model Performance

Model	Average Precision	Average Recall	mAP50-95 (Mean)	Average Inference Time per Image
YOLO	0.88	0.85	0.645	35.92 ms
HOG-SVM	0.97	0.80	-	17.74 ms
SSD	0.23	0.23	0.2314	0.2 ms

YOLO offers a balanced performance with high precision and recall scores (0.88 and 0.85), making it a good fit for applications needing accuracy and real-time capability. Its mAP50-95 score (0.645) is the highest among the models, indicating strong object detection capabilities at varying thresholds. However, it has a longer inference time than SSD and HOG-SVM, at 35.92 ms.

HOG-SVM achieves the highest precision (0.97), making it highly reliable for correct detections, although its recall is slightly lower at 0.80. With an average inference time of 17.74 ms, it performs faster than YOLO but without a comparable mAP metric. This suggests it could be effective for simpler, high-precision tasks with fewer detection thresholds.

SSD is exceptionally fast, with an inference time of only 0.2 ms per image. However, it trades off accuracy for speed, as reflected in its low precision and recall (0.23 each) and mAP50-95 (0.2314). This model may best suit applications where speed is prioritized over high accuracy.

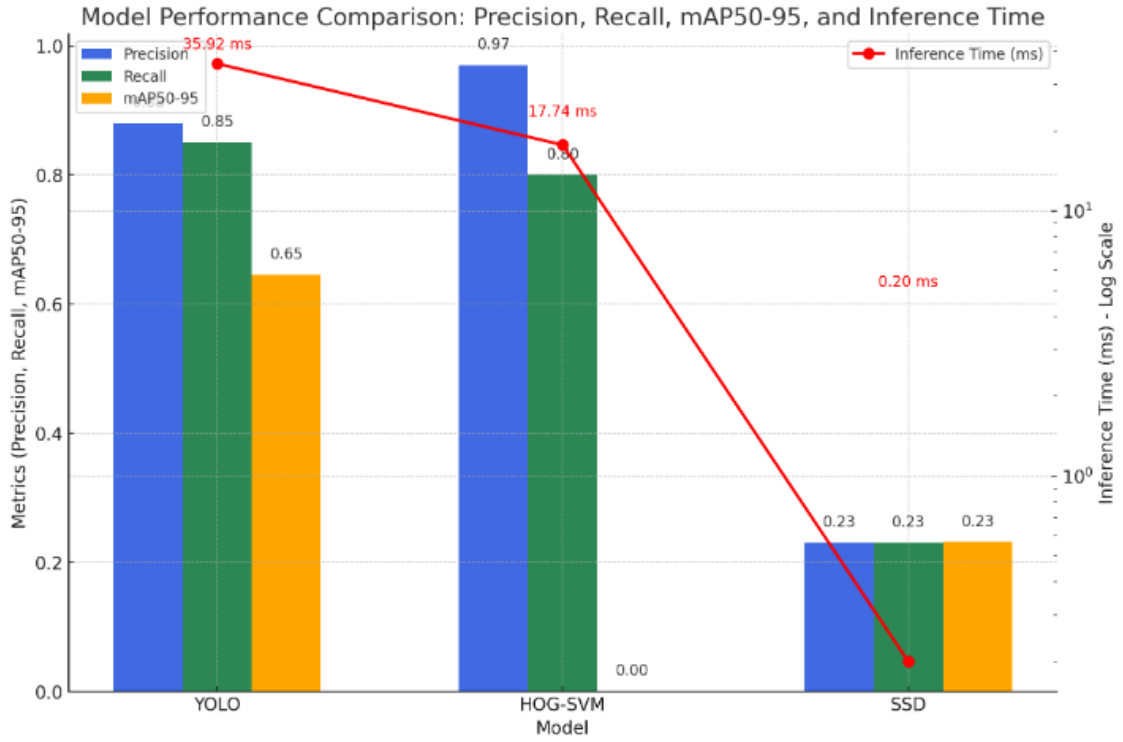


Figure 4. Model Performance Comparison

HOG-SVM demonstrates the highest precision (0.97) and fairly high recall (0.80), indicating it has very few false positives and is generally effective at correctly identifying

objects. YOLO maintains a good balance with both high precision (0.88) and recall (0.85), suggesting it is reliable for applications that require both accuracy and recall. SSD shows lower precision and recall (0.23 each), which limits its suitability for tasks demanding high accuracy but makes it viable for high-speed scenarios with lower accuracy requirements.

YOLO achieves the highest mAP50-95 (0.645), reflecting its accuracy across various thresholds and confirming it as the most robust model for detailed detections. SSD has a significantly lower mAP50-95 (0.2314), reinforcing its trade-off of accuracy for speed. The lack of an mAP score for HOG-SVM suggests it may not be optimized for multi-threshold detections.

SSD is by far the fastest, with an average inference time of 0.2 ms per image, making it ideal for real-time applications where speed is critical. HOG-SVM also performs quickly, averaging 17.74 ms per image, providing a balanced solution between speed and accuracy. YOLO has the longest inference time (35.92 ms), which is still suitable for many real-time applications but might lag slightly compared to SSD and HOG-SVM.

The graph and table underscore each model's strengths; YOLO balances accuracy and speed. It has strong detection capability (highest mAP50-95) but a slower inference time than SSD. HOG-SVM achieves high accuracy, especially in precision, and performs faster than YOLO, making it suitable for high-precision, moderate-speed tasks. SSD prioritizes speed, making it ideal for applications requiring rapid detection but where high accuracy is less critical.

Table 11. Model Accuracies

Model	Overall Accuracy (mAP or Average)
YOLO	92%
HOG-SVM	100% (Training) / 67.76% (Val/Test)
SSD	23.14% (mAP 50-95)

YOLO stands out with a high overall accuracy of 92%, indicating reliable performance in detecting objects across different classes. This level of accuracy makes it suitable for general-purpose object detection tasks that require precision and recall.

HOG-SVM achieves a perfect training accuracy of 100%, showing that it can effectively classify objects it has seen during training. The dramatic drop to 16.67% for

validation accuracy and 20% for test accuracy signals overfitting, where the model memorizes training data but struggles with new, unseen data. This limits HOG-SVM's real-world applicability despite its high precision on the test set.

With a mAP50-95 of 23.14%, SSD demonstrates modest accuracy, which aligns with its purpose as a high-speed model for applications where rapid detection is prioritized over high accuracy. While it may not detect every instance accurately, its speed compensates when real-time performance is critical.

Determining the Best Evaluation Metrics for Each Model

Mean Average Precision (mAP)

Mean Average Precision (mAP) is an important metric for assessing object detection models. It averages precision across different recall levels and classes, typically calculated at specific Intersection over Union (IoU) thresholds, such as mAP@0.5 or mAP@[0.5:0.95]. A higher mAP indicates better performance in accurately detecting true objects.

mAP is particularly relevant for models like YOLO and SSD, which are designed for bounding box detection. The mAP@0.5 threshold is often used in applications that require precise localization, emphasizing the trade-off between accuracy and speed, especially since SSD focuses more on speed.

For simpler models like HOG-SVM, which do not include precise bounding box regression, mAP may be less relevant but can still provide a reference if bounding box predictions are added. Overall, mAP is a key tool for evaluating the effectiveness of object detection models in various scenarios.

Precision and Recall

Precision measures the accuracy of detected objects by calculating the ratio of true positives to all detected objects. At the same time, recall assesses the ability to identify actual objects by comparing true positives to all actual objects.

These metrics are crucial in scenarios where false positives and false negatives have different impacts. High precision is important when false alarms are costly, whereas high recall is necessary when missing detections are unacceptable.

For models like YOLO and SSD, there is often a trade-off between precision and recall. YOLO is strong in recall, capturing most objects, while SSD provides a balance of

moderate precision and recall due to its focus on speed. Analyzing precision and recall helps identify which model suits specific application needs.

In contrast, HOG-SVM usually shows high precision but low recall. Here, precision-recall curves can effectively highlight the model's weaknesses, especially regarding recall.

F1-Score

The F1-score is the harmonic mean of precision and recall, effectively balancing these two metrics into a single value. This score is especially useful when there is a significant disparity between false positives and false negatives. This provides a comprehensive evaluation for models that may excel in precision while struggling with recall or vice versa. When these metrics are imbalanced, it offers a clearer picture of overall performance.

In the context of object detection models like YOLO and SSD, the F1-score serves as an important indicator of how well these models balance false positives and false negatives. For HOG-SVM, which often faces challenges maintaining high recall in unseen data, the F1-score can highlight the extent to which the model's inability to detect all objects compromises its high precision. This dual insight is valuable for assessing and improving model reliability.

Inference Time

Inference time indicates how long a model takes to predict a single image. In real-time applications, faster inference is essential. A quicker detection model allows for rapid responses, even if it sacrifices some accuracy, making this metric important for assessing practical usability.

Both YOLO and SSD are built for real-time detection, with speed being a crucial factor. YOLO is generally faster, while SSD aims to balance accuracy and speed, making it suitable for medium to high-speed needs. HOG-SVM tends to be slower and is less suitable for real-time applications, highlighting its limitations in scenarios where speed is important. It is better suited for applications where speed is not a critical factor.

Accuracy

Accuracy in object detection measures the proportion of correctly classified objects, including their bounding boxes, relative to the total number of objects.

While accuracy can provide some insight, it is generally less meaningful than metrics like Mean Average Precision (mAP) in this context. However, in scenarios where overall correctness is acceptable, accuracy can still be a useful measure.

For models like YOLO and SSD, accuracy can serve as a rough benchmark, but metrics such as mAP and F1-scores offer deeper insights into performance. In contrast, HOG-SVM, which lacks advanced object localization, may have its performance better reflected through accuracy in simpler detection tasks. However, precision and recall metrics will likely provide a more comprehensive understanding of its effectiveness.

Mean IoU (Intersection over Union)

Mean IoU quantifies the overlap between predicted bounding boxes and ground truth boxes. It is commonly used alongside precision and recall when calculating mean Average Precision (mAP). This metric is valuable because it evaluates a model's ability to accurately locate objects within an image, which is essential for performance in models like YOLO and SSD.

For YOLO and SSD, Mean IoU is highly relevant as it indicates how well these models can localize objects with their bounding boxes. A high IoU score signifies precise bounding box placement around detected objects. In contrast, for HOG-SVM, Mean IoU is less applicable since this model is not primarily designed for complex object localization tasks.

Table 12. Summary of Key Metrics for Each Model

Metric	**YOLO**	**HOG-SVM**	**SSD**
mAP	Essential	Less Relevant	Essential
Precision	Important	Very Important	Important
Recall	Important	Very Important	Important
F1-Score	Useful	Useful	Useful
Inference Time	Critical	Moderate	Critical
Accuracy	Moderate Utility	Moderate Utility	Moderate Utility

Mean IoU	Critical	Less Relevant	Critical

In summary, the optimal performance metrics for object detection models vary depending on their design and application context.

For models like YOLO and SSD, the most comprehensive metrics include mean Average Precision (mAP), Intersection over Union (IoU), and the F1-score. These metrics provide a well-rounded assessment of accuracy, while inference time is critical for applications requiring real-time detection.

On the other hand, HOG-SVM benefits most from metrics such as precision, recall, and the F1-score, particularly because it may encounter difficulties with unseen data. Although HOG-SVM is not optimized for precise bounding box localization, accuracy metrics remain relevant for evaluating its performance on simpler detection tasks. Thus, selecting the right metrics tailored to each model's strengths and weaknesses is essential for effective evaluation.