Cover Sheet	
Name	Syed Wajahat Quadri
User ID	W21043564
Course	BSc (Hons) Computer Science with AI
Module Code	KF5042
Module Title	Intelligent Systems
Submission Date	To be submitted by 17:00 GMT on 08/May/2023
Word Count	2750 words

Comparative Analysis of YOLOv2, YOLOv3, and SSD for Human Detection in Surveillance Systems

Syed Wajahat Quadri (W21043564)

Department of Computer Science

Northumbria University

Newcastle Upon Tyne, United Kingdom

syed.w.quadri@northumbria.ac.uk

***Abstract* – Surveillance Systems have become an essential component of security and are being utilized in restricted areas such as governmental buildings, airports, business premises, and even residential homes to identify suspicious activities. Accurate identification of a suspicious activity requires utilization of Object Detection Algorithms. This paper presents a comparative study between YOLOv2, YOLOv3, and SSD that are state-of-the-art (SOTA) object detectors in the surveillance industry. The aim is to identify individuals in various regions while evaluating its Precision-Recall Curve, Average Precision (AP), and Log Average Miss Rate (LAMR). This study resulted in YOLOv2 exceeding the other two models in terms of AP and LAMR.**

Keywords – CNN, Deep Learning, Ethical, Evaluation, Machine Learning, Object Detection, Surveillance, Security, SSD, YOLO.

I. INTRODUCTION

With the alarming rise in crime rates across the world, surveillance systems have become a key instrument in ensuring public safety and security. Surveillance cameras are installed in public places to detect any suspicious activity and to help prevent dangerous incidents from occurring.

Object-detection algorithms play an important role in detecting visual objects from digital media [1] and are used by surveillance systems to detect potential criminal offenses in real-time. The technological evolution in computer vision and machine learning over the past decade has made surveillance systems exponentially powerful [2]. One of the SOTA object detection models in the surveillance sector is the You Only Look Once (YOLO) model. YOLO has shown excellent performance in object detection and is not only used in surveillance systems, but also in other domains such as autonomous vehicles, robotics, medical imaging and sports analysis (more about YOLO in section II).

This study presents an analysis of two YOLO models and SSD model for surveillance systems to detect people. Particularly, the performance of the model in terms of speed, accuracy, and its computational resource consumption as opposed to its variants. The aim of this study is to necessitate effective surveillance systems in order to prevent crimes and to elect the most suitable model out of the three detectors.

The rest of the paper is organized as follows: Section II presents a thorough overview of the domain and the description of YOLOv2 [4], YOLOv3 [5], and SSD [6]. Section III describes the methodology used in this study to achieve comparison functionality. Section IV discusses the results of the conducted study. Section V presents the conclusion and discusses as approach to extend this study for future improvements in the field.

II. RELATED WORK

A. Object-Detection Algorithms

CNNs were crucial in the development of YOLO, an efficient object-detection system proposed by Redmon et al. [1] in 2016. YOLO showcases a unique object-detection approach that uses a CNN to detect objects in an image. The input image is resized and passed through a single CNN. Finally, the model predictions are returned along with the probability percentage [3]. Unlike two-step detectors, YOLO uses a single CNN to concurrently predict the various objects in the image. This can be beneficial in situations where speed is of the utmost importance. To comprehend how YOLO works, it is necessary to understand how the first version of YOLO works.

YOLO: YOLO [3] consists of 24 convolutional layers with 2 fully connected layers at the end. 1×1 reduction layer and 3×3 convolutional layers are used in place of inception modules. Finally, the output layer consists of 7×7×30 prediction tensors. ImageNet 1000-class dataset is used to pretrain the first 20 convolutional layers, the average-pooling layer, and the fully connected layer. The network is then trained for a week on 224×224 pixels input resolution to achieve 63.4 mAP (see Table 1) on PASCAL VOC 2007+2012 and 88% accuracy on ImageNet 2012 validation set. The convolutional layers extract features from the input image of 488×488 pixels while the fully connected layers make predictions of the object, its class probabilities, and its bounding box coordinates [3]. The network is shown in Figure 1.

YOLOv2: YOLOv2 [4] released in 2016 has various improvements over its predecessor (See Fig. 3) and can identify over 9000 categories. Batch normalization is introduced which leads to faster training and mAP increase of 2%. The fully connected layers are removed and the predictions of bounding boxes are made by anchor boxes which increases mAP by 4%. The classifier is trained on 488×488 pixels and the input resolution (resizable) is changed to 416×416 pixels . It uses a joint training algorithm to train for detection and classification. Datasets used are COCO detection dataset and ImageNet-1000 classification dataset. The network is built on Darknet-19 model that contains 19 convolutional layers and 5 maxpooling layers. To train for detection, the last convolutional layer is replaced by three 3×3 convolutional layers each with 1024 filters followed by a 1×1 convolutional layer for output [4].

YOLOv3: YOLOv3 [5] uses Darknet-53 (see Fig. 2) backbone which consists of 53 convolutional layers which contain skip connections and 3 prediction heads (Large, Medium, & Small Scale Detection Head). Although it is slightly slower than YOLOv2 [12], it is known for its speed-accuracy balance. With an AP of 31 (See Table 1) on COCO dataset and having similar accuracy to a two-stage detector Faster R-CNN [14], it makes predictions at three scales using a technique called Feature Pyramid Network (FPN) by downsampling the input image dimensions by 32, 16, and 8. Non-Max Suppression (NMS) is used to remove overlapping bounding boxes and keeping the most confident ones. Moreover, it is better at detecting small objects, which was always a drawback in previous versions [5].

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2. YOLOv3's Darknet-53 [5]

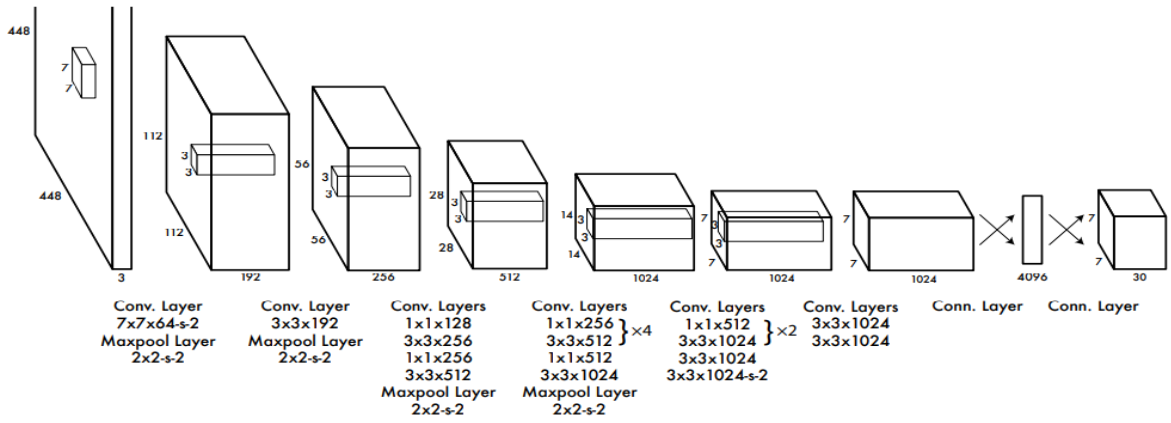

Figure 1. YOLO Network Architecture consisting of 24 convolutional layers and 2 fully connected layers [3]

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 3. Major techniques used by YOLOv2 to improve accuracy over YOLO [4]

Single Shot Multibox Detector (SSD): An object detection model developed in 2016 by researchers at Google [6]. Although it does not resample features for bounding boxes like Faster R-CNN [14] does, it is equally accurate while being significantly faster [6]. An mAP of 74.3% on VOC2007 (See Table 1) is achieved by predicting bounding boxes and object classes using a convolutional filter (3×3) while using separate filters for different image resolutions, and then performing detection at multiple scales by applying the filters to various feature maps [6, 36]. The base network is a CNN, typically VGG-16 or ResNet-50 pretrained on ImageNet that extracts feature maps from the input image of 300×300 pixels [15]. As these features are passed through the network, they are downsampled to detect objects at different scales [6]. As shown in Figure 5, the leftmost base layer (Conv4_3) detects objects at the smallest scale, while the rightmost at the largest scale [15]. If the IOU of bounding boxes is over 0.5, SSD considers it as a positive match. Otherwise, it is considered negative [15]. In Figure 4, the dog (in red) is only detected by the 4×4 feature map layer while the cat (in blue) is only detected by 8×8 feature map layer [6].

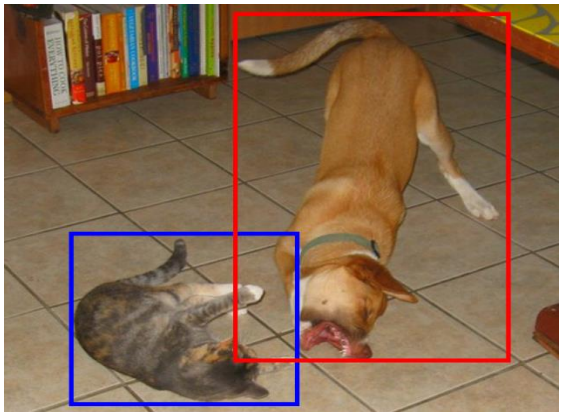

Figure 4. Cat detected by 8×8 feature map and dog detected by 4×4 feature map [6]

B. Real-World Applications in Surveillance Systems

YOLO has been used for intrusion detection, crowd monitoring, traffic monitoring, and facial recognition, all in real-time. In a paper by Warsi et al. [7], YOLOv3 is trained to detect handguns and alert authorities. The study concluded with YOLOv3 performing better than Faster R-CNN [14] in terms of speed while being identical in accuracy. Another study by Jiang et al. [8] used YOLOv3 to detect face masks in real-time during the COVID-19 pandemic. This further proves that YOLOv3 is capable of detecting objects in real-time.

A paper by Ning et al. [16] uses a simplified backbone of SSD called L-SSD for an intelligent surveillance system (ISS) that structures video footage and stores target data. Another paper by Li et al. [17] uses the conventional SSD model and replaces VGG-16 with EfficientNet-B3 as its backbone. Known as EffcientSSD, it is used as a real-time railway perimeter intrusion detector for railway security [17].

Name	Year	Train Dataset	mAP	FPS
Fast R-CNN	2015	VOC 2007 + 2012	70.0	0.5
YOLO			63.4	45
Faster R-CNN (ResNet)			73.2	7
YOLOv2 (416×416)	2016		**76.8**	**67**
SSD		VOC 2007	**74.3**	**59**
YOLOv3 (416×416)	2018	COCO	**31**	**35**
YOLOv4 (416×416)	2019	MS COCO	41.2	38

Table 1. Comparison of YOLOv2, YOLOv3, & SSD [1, 5, 6, 12-14].

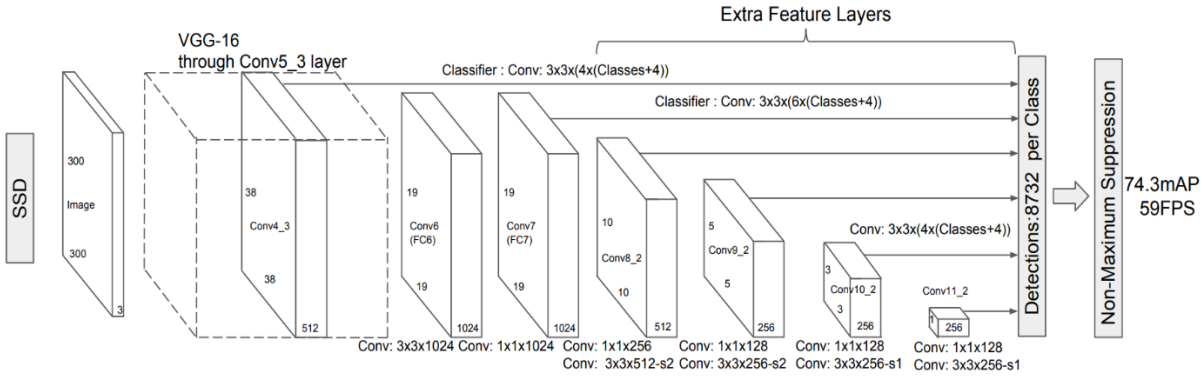

Figure 5. SSD Architecture [6]

C. Ethical, Social, and Legal Review

Surveillance systems have raised ethical concerns amongst those who believe their liberty and privacy is being suppressed while others argue it helps fight crimes [9]. There are individuals that claim the system is overrated and is only effective in some situations [10]. Klauser [11] classifies video surveillance into preservative and protective. Preservative is when it is used to preserve social order, while protective is when it is used to create a safe public environment when necessary. Since modern crime is complex with severe effects, it requires modern solutions to fight said crimes.

III. METHODS

A. Experimental Pipeline

As shown in Figure 6, this section conducts experiments by testing YOLOv2 [4], YOLOv3 [5], and SSD [6]. The experiments will be conducted on MATLAB R2023a using an Nvidia RTX 2060 (Mobile) GPU.

Pedestrian Detection Dataset published by Victor [18] contains images of people in various public locations and is trained and tested on each model to acquire their respective performance statistics. The dataset initially containing 3454 images is shrunk down to 500 images for training and 50 images for testing since all images are manually labelled using Image Labeler on MATLAB, making the data limited. Each training image has 2-8 variations (flipped and slightly rotated), has a resolution of 488×488px, and is 20-80KB in size, making it computationally easier to train the models. All training images are resized and thoroughly shuffled before the training process. Once the models are successfully trained, they are tested on Precision-Recall Curve, Average Precision (AP), and Log Average Miss Rate (LAMR).

B. Evaluation Metrics

In object detection, there are various evaluation metrics that can be utilized. In this study, the following metrics are used for evaluation:

Precision x Recall Curve: The ability of a model to accurately measure positive predictions [23-25] is called precision. In other words, the proportion of true positives (TP) amongst total positive (TP + FP) predictions:

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

The ability of the model to correctly identify true positives (TP) [23-25] is called Recall. In other words, the proportion of true positives (TP) against all ground truths (TP + FN):

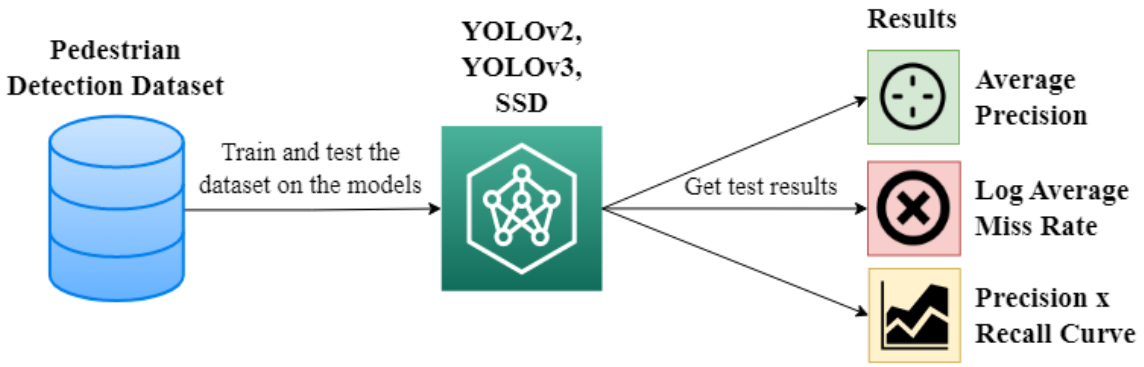

Figure 6. Steps for testing the Surveillance System

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

The curve is created with Precision at the y-axis and Recall at the x-axis. It is a significant test for surveillance systems as it is said to be a good object detection model if the prediction stays high as recall increases [19]. It is a commonly used method of evaluating model performance.

Average Precision (AP): It is the area under the precision x recall curve. The larger the area, the better the machine learning model [26, 27]. It is a useful metric for surveillance systems to detect individuals as a high-accuracy model is necessary to successfully detect criminal activities, identify criminals, and monitor public areas for public safety. The mathematical formula is as follows:

$$AP = \int_0^1 p(r)dr$$

Log Average Miss Rate (LAMR): It refers to the objects that are not detected or missed during detection [21]. It is a graph created by comparing the log average miss rate of the test results to the ground truth [20]. It is an excellent metric to evaluate model accuracy for surveillance systems since it considers the accuracy and completeness of the model. The LAMR value ranges from 0-1, the lower indicating good performance whilst the higher indicating otherwise. The mathematical formula is as follows:

$$LAMR = \exp\left(\frac{1}{9}\sum_f log\begin{pmatrix} mr(argmax\ fpi\ (c)) \\ fppi(c) \le f \end{pmatrix}\right)$$

C. Algorithm Selection

The algorithms chosen for this study are YOLOv2 [4], YOLOv3 [5], and SSD [6]. The reason being that they are single-shot detectors and are known for their fast performance. Hence, they are being tested for accuracy rather than speed to elect the most suitable model for surveillance systems.

YOLOv2 [4] allows a minimum input size of 244×244px and has 7 anchors for detection purposes. ResNet-50 is used as a feature extraction network and the activation function used is ReLU that is applied to the 40th layer of the network. Adaptive Moment Estimation (ADAM) is the optimization algorithm used for this model as it is known for its adaptiveness, efficient weights optimization, and its ability to train large complex datasets. The Mini-Batch is divided into 6 images since the training data is limited and the network size is too large to fit into memory (Although it is best practice to set the mini-batch size to 2^x where x is the number of choice, 6 was chosen since it outputted the best results). The learning rate is set to 0.0001 to ensure gradual convergence of the optimization algorithm. Since the training data is limited and time is of essence, 20 epochs are chosen for the model.

YOLOv3 [5] has a SqueezeNet base that is also pretrained on ImageNet dataset and can identify around 1000 object classes [22]. The feature network extraction in SqueezeNet contains two detection heads, "fire5-concat" and "fire9-concat", the former being twice the size of the latter, making it better for detecting smaller objects [22]. It has an input size of 227×227px and 6 anchors to make detections. The epochs are set to 20 with a learning rate of 0.001. The Mini-Batch size of 6 was found optimal for the model. For the first 700 iterations, the following formula is used to gradually increase the learning rate to stabilize the gradients [22]:

$$LearningRate = \left(\frac{iteration}{warmupPeriod}\right)^4$$

As for the optimization algorithm, Stochastic Gradient Descent with Momentum (SGDM) is used as it is the best for finding the global minimum of gradient zero. L2Regularization, a technique to prevent overfitting is set to 0.0005 and the penalty threshold is set to 0.5 for balanced optimization.

SSD [6] allows 300×300px for input images and uses ResNet-50 as the base network. It uses SGDM

Model	*Input Size*	*Epochs*	*Learning Rate*	*Batch Size*	*Optimization Algorithm*	*Base*
YOLOv2	[224 224 3]	20	0.0001	6	adam	ResNet-50
YOLOv3	[227 227 3]	20	0.001	6	sgdm	SqueezeNet
SSD	[300 300 3]	15	0.001	8	sgdm	ResNet-50

Table 2. Parameters of YOLOv2, YOLOv3, & SSD [1, 5, 6, 12, 13].

algorithm for optimization, mini-batch size of 8, with a learning rate of 0.001 that stays constant throughout training. Only 15 epochs are used for training the model since the data is limited and a high learning rate is being used. Moreover, each epoch is shuffled to reduce bias, enhance generalization, avoid local optima, and to increase the overall robustness of the model.

IV. RESULTS & DISCUSSION

A. Performance Analysis of Models

Figure 7 and 8 display the comparative results of the MATLAB evaluation metrics between the models. The average precision of YOLOv2 [4] and SSD [6] is 0.7, while YOLOv3 [5] falls behind with 0.65. Although YOLOv3 [5] is known for its better accuracy over its predecessors, it fails to outperform YOLOv2 [4] in this experiment. This may be due to the larger network architecture of the YOLOv3 [5] model, compared to YOLOv2 [4] and SSD [6] which are smaller models. Additionally, YOLOv3 has the highest LAMR of 0.62 while YOLOv2 [4] is 0.57 and SSD [6] is 0.59. This means that YOLOv3 [5] misses a lot more target objects than YOLOv2 and SSD [6]. Moreover, according to the FPS counter in Table 1, YOLOv3 [5] has an FPS of 35 while YOLOv2 [4] has 67 and SSD [6] has 59. Hence, YOLOv3 [5] is considered unreliable for surveillance systems according to the findings of this experiment.

While YOLOv2 [4] and SSD [6] have the same AP of 0.7, YOLOv2 [4] manages to outperform SSD [6] with a 0.02% decrease in LAMR. YOLOv2 [4] also outputs higher frames (See Table 1) with 67 FPS while SSD [6] outputs 59 FPS, making it perfect for quick detections. Moreover, YOLOv2 [4] is a lot easier and faster to train, giving it the edge over SSD [6].

B. Ethical Issues of predictive performance

Use of object-detection models to monitor people may be seen as intrusive, especially when done without their knowledge. Moreover, the detection models may be prone to biases, leading to unfair profiling of individuals. This may result in social inequalities.

The model might misidentify individuals (false positives) leading to false accusations and wrongful arrests. Furthermore, the model could be used for reasons unrelated to public safety, leading to infringement of civil liberties.

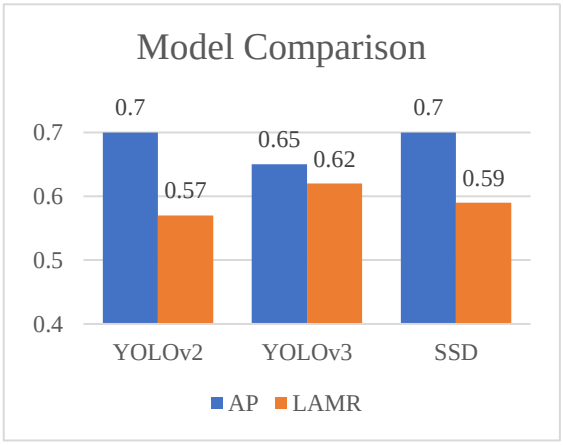

Figure 8. *AP and LAMR of all models*

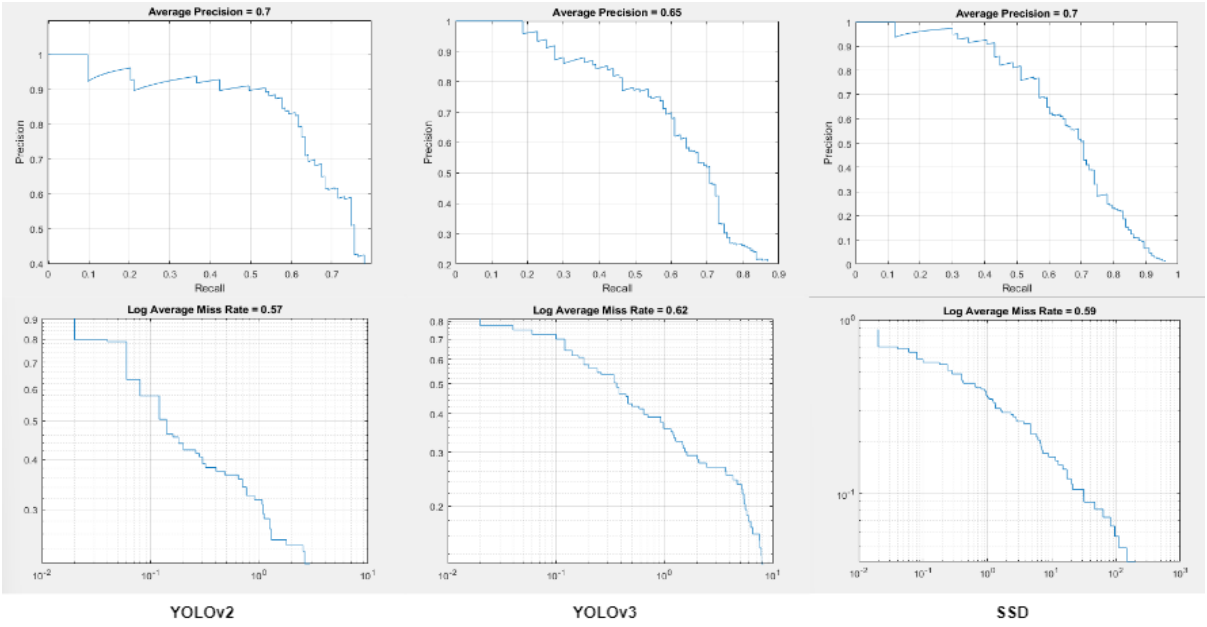

Figure 7. Precision x Recall Curve, AP, and LAMR of YOLOv2, YOLOv3, and SSD

V. CONCLUSION & FUTURE WORK

YOLOv2 [4], YOLOv3 [5], and SSD [6] were introduced and their networks were explained. The models were trained and tested to elect the most suitable model for surveillance systems to detect individuals. It is necessary for surveillance systems to detect individuals accurately to maintain public safety. YOLOv2 [4] scored the highest AP, the lowest LAMR (see Fig. 8), and its Precision x Recall Curve is the furthest to the right out of the three models (see Fig. 7). Hence, this paper proposes that YOLOv2 [4] is the most optimal option for surveillance systems due to the model's high accuracy, low miss rate, and its ability to train and perform in low-performing environments.

For future work, it is recommended to use similar techniques to further minimize LAMR and increase AP to maximize the robustness of the model for surveillance systems.

VI. REFERENCES

1. Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," Proceedings of the IEEE, 2023. [Online]. Available:https://ieeexplore.ieee.org/abstract/document/10028728. [Accessed: April 13, 2023].
2. M. Valera and S. A. Velastin, "Intelligent distributed surveillance systems: a review," IEE Proceedings-Vision, Image and Signal Processing, vol. 152, no. 2, pp. 192-204, Apr. 2005.
3. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.
4. J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 6517-6525.
5. J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 6517-6527, doi: 10.48550/arXiv.1804.02767
6. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, and A.C. Berg, "Ssd: Single shot multibox detector," in Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14, pp. 21-37, Springer International Publishing, 2016. Available: https://arxiv.org/pdf/1512.02325.pdf
7. A. Warsi, M. Abdullah, M.N. Husen, M. Yahya, S. Khan, and N. Jawaid, "Gun detection system using YOLOv3," in 2019 IEEE International Conference on Smart Instrumentation, Measurement and Application (ICSIMA), Aug. 2019, pp. 1-4.
8. Jiang, X., Gao, T., Zhu, Z., and Zhao, Y. (2021). "Real-time face mask detection method based on YOLOv3," Electronics, 10(7), 837.
9. A. Sutton and D. Wilson, "Open-Street CCTV in Australia: The Politics of Resistance and Expansion," Surveillance and Society, vol. 2, no. 2/3, pp. 187-206, Jan. 2004
10. S. Graham, "CCTV: The Stealthy Emergence of a Fifth Utility?," Planning Theory and Practice, vol. 3, no. 2, pp. 237-241, Jun. 2002.
11. F. Klauser, "A Comparison of the Impact of Protective and Preservative Video Surveillance on Urban Territoriality: the Case of Switzerland," Surveillance and Society, vol. 2, no. 3, pp. 145-160, Sep. 2004.
12. A. Kathuria, "What's new in YOLO v3?," Medium, Apr. 29, 2018. https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
13. V. Praharsha, "YOLOv4 model architecture," OpenGenus IQ: Computing Expertise & Legacy, Jan. 11, 2022. https://iq.opengenus.org/yolov4-model-architecture/
14. S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems, vol. 28, pp. 91-99, 2015.
15. J. Hui, "SSD object detection: Single Shot MultiBox Detector for real-time processing," Medium, Dec. 28, 2018. https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06
16. K. Ning et al., "An Intelligent Surveillance System Based on Lightweight Object Detection Network L-SSD," in 2019 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA), 2019, pp. 156–157. doi: 10.1109/ICTA48799.2019.9012867.
17. Y. Li et al., "Efficient SSD: A Real-Time Intrusion Object Detection Algorithm for Railway Surveillance," in 2020 International Conference on Sensing, Diagnostics, Prognostics, and Control (SDPC), 2020, pp. 391–395. doi: 10.1109/SDPC49476.2020.9353137.

18. Victor Reg, PedestriansDetection Dataset . Roboflow , 2022. [Online]. Available: %20https://universe.roboflow.com/victor-reg/pedestriansdetection%20
19. V. Dubey, “Evaluation Metrics for Object detection algorithms,” Medium, Oct. 06, 2020. https://medium.com/@vijayshankerdubey550/evaluation-metrics-for-object-detection-algorithms-b0d6489879f3
20. “Evaluate miss rate metric for object detection - MATLAB evaluateDetectionMissRate - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ref/evaluatedetectionmissrate.html#bvinr0i-1-fppi (accessed May 02, 2023).
21. P. Dollar, C. Wojek, B. Schiele and P. Perona, "Pedestrian Detection: An Evaluation of the State of the Art," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 4, pp. 743-761, April 2012, doi: 10.1109/TPAMI.2011.155.
22. “Object Detection Using YOLO v3 Deep Learning - MATLAB & Simulink - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ug/object-detection-using-yolo-v3-deep-learning.html (accessed May 04, 2023).
23. J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in Proceedings of the 23rd International Conference on Machine Learning, Jun. 2006, pp. 233-240.
24. C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and F-score, with implication for evaluation," in Advances in Information Retrieval: 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005. Proceedings 27, Springer Berlin Heidelberg, 2005, pp. 345-359.
25. D. M. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," arXiv preprint arXiv:2010.16061, 2020.
26. P. Huilgol, “Precision and Recall | Essential Metrics for Data Analysis (Updated 2023),” Analytics Vidhya, Sep. 03, 2020. https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/#What_is_Precision? (accessed Apr. 23, 2023).
27. D. Cochard, “mAP : Evaluation metric for object detection models,” axinc-ai, Oct. 06, 2021. https://medium.com/axinc-ai/map-evaluation-metric-of-object-detection-model-dd20e2dc2472
28. “Object Detection Using YOLO v2 Deep Learning - MATLAB & Simulink - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/deeplearning/ug/object-detection-using-yolo-v2.html
29. “Getting Started with YOLO v2 - MATLAB & Simulink - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ug/getting-started-with-yolo-v2.html (accessed May 10, 2023).
30. “Train YOLO v2 object detector - MATLAB trainYOLOv2ObjectDetector - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ref/trainyolov2objectdetector.html (accessed May 10, 2023).
31. “Detect objects using YOLO v2 object detector - MATLAB - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ref/yolov2objectdetector.html (accessed May 10, 2023).
32. “Object Detection Using YOLO v3 Deep Learning - MATLAB & Simulink - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ug/object-detection-using-yolo-v3-deep-learning.html
33. “Getting Started with YOLO v3 - MATLAB & Simulink - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ug/getting-started-with-yolo-v3.html (accessed May 10, 2023).
34. “Detect objects using YOLO v3 object detector - MATLAB - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ref/yolov3objectdetector.html (accessed May 10, 2023).
35. “Object Detection Using SSD Deep Learning - MATLAB & Simulink - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ug/object-detection-using-single-shot-detector.html (accessed May 10, 2023).
36. “Getting Started with SSD Multibox Detection - MATLAB & Simulink - MathWorks United Kingdom,” uk.mathworks.com. https://uk.mathworks.com/help/vision/ug/getting-started-with-ssd.html (accessed May 10, 2023).
37. “Train an SSD deep learning object detector - MATLAB trainSSDObjectDetector - MathWorks United Kingdom,” uk.mathworks.com.

https://uk.mathworks.com/help/vision/ref/trainssdobjectdetector.html (accessed May, 2023).

VII. ADDENDUM

Repository link to code files – https://github.com/swq1999/matlab-assessment-code

A. YOLOv2 Code

```
clear; clc;

%-------------------------- LOAD TRAINING DATA ------------------------

myData = load('Pedestrians\train\trainLabels.mat');
names = myData.gTruth.DataSource.Source(:,1);
person =  myData.gTruth.LabelData.person;
Dataset = table(names, person);

% Shuffling train data
rng(0);
shuffledIdx = randperm(height(Dataset));
trainingData = Dataset(shuffledIdx,:);

% Image datastore
imds = imageDatastore(trainingData.names);

% label datastore
blds = boxLabelDatastore(trainingData(:,2:end));

% combine
ds = combine(imds,blds);

% Validating data
validateInputData(ds);

% Minimum Input Size
ImgInputSize = [224 224 3];
numOfClasses = width(Dataset)-1;
estimationData = transform(ds,@(data)preprocessData(data,ImgInputSize));
anchors = 7;
[anchorBoxes, meanIoU] = estimateAnchorBoxes(estimationData, anchors);
fNetwork = resnet50;
fLayer = 'activation_40_relu';
lgraph =
yolov2Layers(ImgInputSize,numOfClasses,anchorBoxes,fNetwork,fLayer);
augmentedTrainData = transform(ds,@augmentData);

% Augmenting & Visualizing Images
myAugmentedData = cell(4,1);
for k = 1:4
    trainData = read(augmentedTrainData);
    myAugmentedData{k} =
insertShape(trainData{1},'rectangle',trainData{2});
    reset(augmentedTrainData);
end

figure
montage(myAugmentedData,'BorderSize',10)
preprocessedData =
transform(augmentedTrainData,@(data)preprocessData(data,ImgInputSize));
trainData = read(preprocessedData);
myImg = trainData{1};
```

```
mybbox = trainData{2};
annotatedImage = insertShape(myImg,'rectangle',mybbox);
annotatedImage = imresize(annotatedImage,2);
figure
imshow(annotatedImage)

% Training options
options = trainingOptions('adam', ...
        'MiniBatchSize', 6, ....
        'InitialLearnRate',0.0001, ...
        'MaxEpochs',20);

  % Train YOLov2
  [YOLOv2Detector,info] =
trainYOLOv2ObjectDetector(preprocessedData,lgraph,options);

%--------------------------- TESTING AN IMG ------------------------

myImg = imread('rider.jpg');
myImg = imresize(myImg,ImgInputSize(1:2));
[person,scores] = detect(YOLOv2Detector,myImg, Threshold=0.6);
myImg = insertObjectAnnotation(myImg,'rectangle',person,scores);
figure
imshow(myImg)

%--------------------- TESTING & EVALUATING METRICS --------------

% Load test dataset
myData2 = load('Pedestrians\test\testLabels.mat');
names = myData2.gTruth.DataSource.Source(:,1);
person =  myData2.gTruth.LabelData.person;
Dataset2 = table(names, person);

% Image datastore
imds2 = imageDatastore('Pedestrians\test');
% label datastore
blds2 = boxLabelDatastore(Dataset2(:,'person'));

% Run Trained Detector
myDetectionResults = detect(YOLOv2Detector, imds2,Threshold=0.1);

% -------------------- Average Precision ----------------------------

[ap,recall,precision] = evaluateDetectionPrecision(myDetectionResults,
blds2);

figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.1f',ap))

% -------------------- Area Under Curve (AUC) -------------------------

x = recall;
y = precision;
auc = trapz(x,y);
```

```
[am, fppi, missRate] = evaluateDetectionMissRate(myDetectionResults,
blds2);

figure;
loglog(fppi, missRate);
grid on
title(sprintf('Log Average Miss Rate = %.2f', am))
```

**Note: The supporting functions are necessary for the main code to function.

Supporting functions for YOLOv2 obtained from MathWorks Website [28-31]:

```
%------------------------ Augment Data -------------------------------
function B = augmentData(A)
% Apply random horizontal flipping, and random X/Y scaling. Boxes that
% get scaled outside the bounds are clipped if the overlap is above
% 0.25. jitter image color.

B = cell(size(A));

I = A{1};
sz = size(I);
if numel(sz)==3 && sz(3) == 3
    I = jitterColorHSV(I,...
        'Contrast',0.2,...
        'Hue',0,...
        'Saturation',0.1,...
        'Brightness',0.2);
end

% Randomly flip and scale image.
tform = randomAffine2d('XReflection',true,'Scale',[1 1.1]);
rout = affineOutputView(sz,tform,'BoundsStyle','CenterOutput');
B{1} = imwarp(I,tform,'OutputView',rout);


% Apply same transform to boxes.
[B{2},indices] = bboxwarp(A{2},tform,rout,'OverlapThreshold',0.25);
B{3} = A{3}(indices);

% Return original data only when all boxes are removed by warping.
if isempty(indices)
    B = A;
end
end

%------------------------ Preprocess Data ----------------------------
function data = preprocessData(data,targetSize)
% Resize image and bounding boxes to the targetSize.
sz = size(data{1},[1 2]);
scale = targetSize(1:2)./sz;
data{1} = imresize(data{1},targetSize(1:2));

% Resize boxes to new image size.
data{2} = bboxresize(data{2},scale);
end
```

```
%------------------------- Validate Input Data -------------------------
function validateInputData(ds)
% Validates the input images, bounding boxes and labels and displays the
% paths of invalid samples.

% Copyright 2021 The MathWorks, Inc.

% Path to images
info = ds.UnderlyingDatastores{1}.Files;

ds = transform(ds, @isValidDetectorData);
data = readall(ds);

validImgs = [data.validImgs];
validBoxes = [data.validBoxes];
validLabels = [data.validLabels];

msg = "";

if(any(~validImgs))
    imPaths = info(~validImgs);
    str = strjoin(imPaths, '\n');
    imErrMsg = sprintf("Input images must be non-empty and have 2 or 3
dimensions. The following images are invalid:\n") + str;
    msg = (imErrMsg + newline + newline);
end

if(any(~validBoxes))
    imPaths = info(~validBoxes);
    str = strjoin(imPaths, '\n');
    boxErrMsg = sprintf("Bounding box data must be M-by-4 matrices of
positive integer values. The following images have invalid bounding box
data:\n") ...
        + str;

    msg = (msg + boxErrMsg + newline + newline);
end

if(any(~validLabels))
    imPaths = info(~validLabels);
    str = strjoin(imPaths, '\n');
    labelErrMsg = sprintf("Labels must be non-empty and categorical. The
following images have invalid labels:\n") + str;

    msg = (msg + labelErrMsg + newline);
end

if(~isempty(msg))
    error(msg);
end

end

function out = isValidDetectorData(data)
% Checks validity of images, bounding boxes and labels
for i = 1:size(data,1)
    I = data{i,1};
    boxes = data{i,2};
    labels = data{i,3};
    imageSize = size(I);
    mSize = size(boxes, 1);
```

```
    out.validImgs(i) = iCheckImages(I);
    out.validBoxes(i) = iCheckBoxes(boxes, imageSize);
    out.validLabels(i) = iCheckLabels(labels, mSize);
end
end

function valid = iCheckImages(I)
% Validates the input images.

valid = true;
if ndims(I) == 2
    nDims = 2;
else
    nDims = 3;
end
% Define image validation parameters.
classes        = {'numeric'};
attrs          = {'nonempty', 'nonsparse', 'nonnan', 'finite', 'ndims',
nDims};
try
    validateattributes(I, classes, attrs);
catch
    valid = false;
end

end

function valid = iCheckBoxes(boxes, imageSize)
% Validates the ground-truth bounding boxes to be non-empty and finite.

valid = true;
% Define bounding box validation parameters.
classes = {'numeric'};
attrs   = {'nonempty', 'integer', 'nonnan', 'finite', 'positive',
'nonzero', 'nonsparse', '2d', 'ncols', 4};
try
    validateattributes(boxes, classes, attrs);
    % Validate if bounding box in within image boundary.
    validateattributes(boxes(:,1)+boxes(:,3)-1, classes, {'<=',
imageSize(2)});
    validateattributes(boxes(:,2)+boxes(:,4)-1, classes, {'<=',
imageSize(1)});
catch
    valid = false;
end
end

function valid = iCheckLabels(labels, mSize)
% Validates the labels.

valid = true;
% Define label validation parameters.
classes = {'categorical'};
attrs   = {'nonempty', 'nonsparse', '2d', 'ncols', 1, 'nrows', mSize};
try
    validateattributes(labels, classes, attrs);
catch
    valid = false;
end
end
```

B. YOLOv3 Code

```
clear; clc;

%---------------------- LOAD TRAINING DATA -----------------------------

myData = load('Pedestrians\train\trainLabels.mat');
names = myData.gTruth.DataSource.Source(:,1);
person =  myData.gTruth.LabelData.person;
Dataset = table(names, person);

% Splitting the dataset into train and test
rng(0);
shuffledIdx = randperm(height(Dataset));
trainingData = Dataset(shuffledIdx,:);

% Image datastore
imds = imageDatastore(trainingData.names);

% label datastore
blds = boxLabelDatastore(trainingData(:,2:end));

% combine
ds = combine(imds,blds);

% Validating data
validateInputData(ds);

% Augmenting data
augmentedData = transform(ds, @augmentData);

% Visualize the augmented images.
myAugmentedData = cell(4,1);
for k = 1:4
    trainData = read(augmentedData);
    myAugmentedData{k} = insertShape(trainData{1,1}, 'Rectangle',
trainData{1,2});
    reset(augmentedData);
end
figure
montage(myAugmentedData, 'BorderSize', 10)

% define network
imgInputSize = [227 227 3];

rng(0)
DataEstimation = transform(ds, @(data)preprocessData(data,
imgInputSize));
numOfAnchors = 6;
[anchors, meanIoU] = estimateAnchorBoxes(DataEstimation, numOfAnchors);

% anchor boxes
myArea = anchors(:, 1).*anchors(:, 2);
[~, idx] = sort(myArea, 'descend');
anchors = anchors(idx, :);
anchorBoxes = {anchors(1:3,:)
    anchors(4:6,:)
    };

% load base network
network = squeezenet;
```

```
className  = {'person'};

% Detector
yolov3Detector = yolov3ObjectDetector(network, className,
anchorBoxes, ...
    'DetectionNetworkSource', {'fire9-concat', 'fire5-concat'},
InputSize = imgInputSize);

% Preprocess training data
preprocessedData = transform(augmentedData,
@(data)preprocess(yolov3Detector, data));
trainData = read(preprocessedData);

% define img
myImg = trainData{1,1};
bbox = trainData{1,2};
annotatedImg = insertShape(myImg, 'Rectangle', bbox);
annotatedImg = imresize(annotatedImg,2);
figure
imshow(annotatedImg)

reset(preprocessedData);

% Training options
epochs = 20;
batchSize = 6;
lRate = 0.001;
warmup = 700;
l2 = 0.0005;
penaltyThres = 0.5;
vel = [];

% Train model
if canUseParallelPool
   dispatchInBackground = true;
else
   dispatchInBackground = false;
end

mbq = minibatchqueue(preprocessedData, 2,...
        "MiniBatchSize", batchSize,...
        "MiniBatchFcn", @(images, boxes, labels) createBatchData(images,
boxes, labels, className), ...
        "MiniBatchFormat", ["SSCB", ""],...
        "DispatchInBackground", dispatchInBackground,...
        "OutputCast", ["", "double"]);

% TRAINING
    % Create subplots for the learning rate and mini-batch loss.
    fig = figure;
    [lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(fig);

    iteration = 0;
    % Custom training loop.
    for epoch = 1:epochs
        reset(mbq);
        shuffle(mbq);
        while(hasdata(mbq))
            iteration = iteration + 1;
```

```
            [XTrain, YTrain] = next(mbq);

            % Evaluate the model gradients and loss using dlfeval and
the
            % modelGradients function.
            [gradients, state, lossInfo] = dlfeval(@modelGradients,
yolov3Detector, XTrain, YTrain, penaltyThres);

            % Apply L2 regularization.
            gradients = dlupdate(@(g,w) g + l2*w, gradients,
yolov3Detector.Learnables);

            % Determine the current learning rate value.
            currentLR = piecewiseLearningRateWithWarmup(iteration,
epoch, lRate, warmup, epochs);

            % Update the detector learnable parameters using the SGDM
optimizer.
            [yolov3Detector.Learnables, vel] =
sgdmupdate(yolov3Detector.Learnables, gradients, vel, currentLR);

            % Update the state parameters of dlnetwork.
            yolov3Detector.State = state;

            % Display progress.
            displayLossInfo(epoch, iteration, currentLR, lossInfo);

            % Update training plot with new points.
            updatePlots(lossPlotter, learningRatePlotter, iteration,
currentLR, lossInfo.totalLoss);
        end
    end

%----------------------- TESTING AN IMG --------------------------------

myImg = imread('34.png');
[person,scores] = detect(yolov3Detector,myImg, Threshold=0.1);
myImg = insertObjectAnnotation(myImg,'rectangle',person,scores);
figure
imshow(myImg)

%-------------------- TESTING & EVALUATING METRICS --------------------

% Load test data
myData2 = load('Pedestrians\test\testLabels.mat');
names = myData2.gTruth.DataSource.Source(:,1);
person =  myData2.gTruth.LabelData.person;
Dataset2 = table(names, person);

% Image datastore
imds2 = imageDatastore('Pedestrians\test');
% label datastore
blds2 = boxLabelDatastore(Dataset2(:,'person'));

% Get results
results = detect(yolov3Detector,imds2,'MiniBatchSize',6,
Threshold=0.01);

% --------------------- Average Precision ----------------------------
```

```
% Evaluate AP
[ap,recall,precision] = evaluateDetectionPrecision(results,blds2);

figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))

% ------------------- Area Under Curve (AUC) -------------------------

x = recall;
y = precision;
auc = trapz(x,y);

% --------------------- Log Average Miss Rate ------------------------

[am, fppi, missRate] = evaluateDetectionMissRate(results, blds2);

figure;
loglog(fppi, missRate);
grid on
title(sprintf('Log Average Miss Rate = %.2f', am))
```

**Note: The supporting functions are necessary for the main code to function.

Supporting functions for YOLOv3 obtained from MathWorks Website [32-34]:

```
%------------------------ Augment Data -------------------------------
function data = augmentData(A)
% Apply random horizontal flipping, and random X/Y scaling. Boxes that
get
% scaled outside the bounds are clipped if the overlap is above 0.25.
Also,
% jitter image color.

data = cell(size(A));
for ii = 1:size(A,1)
    I = A{ii,1};
    bboxes = A{ii,2};
    labels = A{ii,3};
    sz = size(I);

    if numel(sz) == 3 && sz(3) == 3
        I = jitterColorHSV(I,...
            'Contrast',0.0,...
            'Hue',0.1,...
            'Saturation',0.2,...
            'Brightness',0.2);
    end

    % Randomly flip image.
    tform = randomAffine2d('XReflection',true,'Scale',[1 1.1]);
    rout = affineOutputView(sz,tform,'BoundsStyle','centerOutput');
    I = imwarp(I,tform,'OutputView',rout);

    % Apply same transform to boxes.
    [bboxes,indices] =
bboxwarp(bboxes,tform,rout,'OverlapThreshold',0.25);
    bboxes = round(bboxes);
```

```
    labels = labels(indices);

    % Return original data only when all boxes are removed by warping.
    if isempty(indices)
        data(ii,:) = A(ii,:);
    else
        data(ii,:) = {I, bboxes, labels};
    end
end
end

%------------------------ BBox offset loss ----------------------------
function boxLoss = bboxOffsetLoss(boxPredCell, boxDeltaTarget,
boxMaskTarget, boxErrorScaleTarget)
% Mean squared error for bounding box position.
lossX = sum(cellfun(@(a,b,c,d)
mse(a.*c.*d,b.*c.*d),boxPredCell(:,1),boxDeltaTarget(:,1),boxMaskTarget(
:,1),boxErrorScaleTarget));
lossY = sum(cellfun(@(a,b,c,d)
mse(a.*c.*d,b.*c.*d),boxPredCell(:,2),boxDeltaTarget(:,2),boxMaskTarget(
:,1),boxErrorScaleTarget));
lossW = sum(cellfun(@(a,b,c,d)
mse(a.*c.*d,b.*c.*d),boxPredCell(:,3),boxDeltaTarget(:,3),boxMaskTarget(
:,1),boxErrorScaleTarget));
lossH = sum(cellfun(@(a,b,c,d)
mse(a.*c.*d,b.*c.*d),boxPredCell(:,4),boxDeltaTarget(:,4),boxMaskTarget(
:,1),boxErrorScaleTarget));
boxLoss = lossX+lossY+lossW+lossH;
end

%------------------------ Class Confidence loss ------------------------
function clsLoss = classConfidenceLoss(classPredCell, classTarget,
boxMaskTarget)
% Binary cross-entropy loss for class confidence score.
clsLoss = sum(cellfun(@(a,b,c)
crossentropy(a.*c,b.*c,'TargetCategories','independent'),classPredCell,c
lassTarget,boxMaskTarget(:,3)));
end

%-------------- Configure training progress plotter -------------------
function [lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(f)
% Create the subplots to display the loss and learning rate.
figure(f);
clf
subplot(2,1,1);
ylabel('Learning Rate');
xlabel('Iteration');
learningRatePlotter = animatedline;
subplot(2,1,2);
ylabel('Total Loss');
xlabel('Iteration');
lossPlotter = animatedline;
end

%------------------------ Create batch data ---------------------------
function [XTrain, YTrain] = createBatchData(data, groundTruthBoxes,
groundTruthClasses, classNames)
% Returns images combined along the batch dimension in XTrain and
% normalized bounding boxes concatenated with classIDs in YTrain
```

```
% Concatenate images along the batch dimension.
XTrain = cat(4, data{:,1});

% Get class IDs from the class names.
classNames = repmat({categorical(classNames')},
size(groundTruthClasses));
[~, classIndices] = cellfun(@(a,b)ismember(a,b), groundTruthClasses,
classNames, 'UniformOutput', false);

% Append the label indexes and training image size to scaled bounding
boxes
% and create a single cell array of responses.
combinedResponses = cellfun(@(bbox, classid)[bbox, classid],
groundTruthBoxes, classIndices, 'UniformOutput', false);
len = max( cellfun(@(x)size(x,1), combinedResponses ) );
paddedBBoxes = cellfun( @(v) padarray(v,[len-size(v,1),0],0,'post'),
combinedResponses, 'UniformOutput',false);
YTrain = cat(4, paddedBBoxes{:,1});
end

%------------------------ Display loss info ----------------------------
function displayLossInfo(epoch, iteration, currentLR, lossInfo)
% Display loss information for each iteration.
disp("Epoch : " + epoch + " | Iteration : " + iteration + " | Learning
Rate : " + currentLR + ...
    " | Total Loss : " + double(gather(extractdata(lossInfo.totalLoss)))
+ ...
    " | Box Loss : " + double(gather(extractdata(lossInfo.boxLoss)))
+ ...
    " | Object Loss : " + double(gather(extractdata(lossInfo.objLoss)))
+ ...
    " | Class Loss : " + double(gather(extractdata(lossInfo.clsLoss))));
end

%------------------------ Generate targets ----------------------------
function [boxDeltaTarget, objectnessTarget, classTarget, maskTarget,
boxErrorScaleTarget] = generateTargets(YPredCellGathered,
groundTruth,...
    inputImageSize, anchorBoxes, penaltyThreshold)
% generateTargets creates target array for every prediction element
% x, y, width, height, confidence scores and class probabilities.

boxDeltaTarget = cell(size(YPredCellGathered,1),4);
objectnessTarget = cell(size(YPredCellGathered,1),1);
classTarget = cell(size(YPredCellGathered,1),1);
maskTarget = cell(size(YPredCellGathered,1),3);
boxErrorScaleTarget = cell(size(YPredCellGathered,1),1);

% Normalize the ground truth boxes w.r.t image input size.
gtScale = [inputImageSize(2) inputImageSize(1) inputImageSize(2)
inputImageSize(1)];
groundTruth(:,1:4,:,:) = groundTruth(:,1:4,:,:)./gtScale;

anchorBoxesSet = cell2mat(anchorBoxes);

maskIdx = 1:size(anchorBoxesSet,1);
cellsz = cellfun(@size,anchorBoxes,'uni',false);
convMask = cellfun(@(v)v(1),cellsz);
anchorBoxMask = mat2cell(maskIdx,1,convMask)';
```

```
for numPred = 1:size(YPredCellGathered,1)

    % Select anchor boxes based on anchor box mask indices.
    anchors = anchorBoxes{numPred, :};

    bx = YPredCellGathered{numPred,2};
    by = YPredCellGathered{numPred,3};
    bw = YPredCellGathered{numPred,4};
    bh = YPredCellGathered{numPred,5};
    predClasses = YPredCellGathered{numPred,6};

    gridSize = size(bx);
    if numel(gridSize)== 3
        gridSize(4) = 1;
    end
    numClasses = size(predClasses,3)./size(anchors,1);

    % Initialize the required variables.
    mask = single(zeros(size(bx)));
    confMask = single(ones(size(bx)));
    classMask = single(zeros(size(predClasses)));
    tx = single(zeros(size(bx)));
    ty = single(zeros(size(by)));
    tw = single(zeros(size(bw)));
    th = single(zeros(size(bh)));
    tconf = single(zeros(size(bx)));
    tclass = single(zeros(size(predClasses)));
    boxErrorScale = single(ones(size(bx)));

    % Get the IOU of predictions with groundtruth.
    iou = getMaxIOUPredictedWithGroundTruth(bx,by,bw,bh,groundTruth);

    % Donot penalize the predictions which has iou greater than penalty
    % threshold.
    confMask(iou > penaltyThreshold) = 0;

    for batch = 1:gridSize(4)
        truthBatch = groundTruth(:,1:5,:,batch);
        truthBatch = truthBatch(all(truthBatch,2),:);
        % Get boxes with center as 0.
        gtPred = [0-truthBatch(:,3)/2,0-
truthBatch(:,4)/2,truthBatch(:,3),truthBatch(:,4)];
        anchorPrior = [0-anchorBoxesSet(:,2)/(2*inputImageSize(2)),0-
anchorBoxesSet(:,1)/(2*inputImageSize(1)),anchorBoxesSet(:,2)/inputImage
Size(2),anchorBoxesSet(:,1)/inputImageSize(1)];

        % Get the iou of best matching anchor box.
        overLap = bboxOverlapRatio(gtPred,anchorPrior);
        [~,bestAnchorIdx] = max(overLap,[],2);

        % Select gt that are within the mask.
        index = ismember(bestAnchorIdx,anchorBoxMask{numPred});
        truthBatch = truthBatch(index,:);
        bestAnchorIdx = bestAnchorIdx(index,:);
        bestAnchorIdx = bestAnchorIdx - anchorBoxMask{numPred}(1,1) + 1;

        if ~isempty(truthBatch)
            % Convert top left position of ground-truth to centre
coordinates.
```

```
            truthBatch =
[truthBatch(:,1)+truthBatch(:,3)./2,truthBatch(:,2)+truthBatch(:,4)./2,t
ruthBatch(:,3),truthBatch(:,4),truthBatch(:,5)];

            errorScale = 2 - truthBatch(:,3).*truthBatch(:,4);
            truthBatch =
[truthBatch(:,1)*gridSize(2),truthBatch(:,2)*gridSize(1),truthBatch(:,3)
*inputImageSize(2),truthBatch(:,4)*inputImageSize(1),truthBatch(:,5)];
            for t = 1:size(truthBatch,1)

                % Get the position of ground-truth box in the grid.
                colIdx = ceil(truthBatch(t,1));
                colIdx(colIdx<1) = 1;
                colIdx(colIdx>gridSize(2)) = gridSize(2);
                rowIdx = ceil(truthBatch(t,2));
                rowIdx(rowIdx<1) = 1;
                rowIdx(rowIdx>gridSize(1)) = gridSize(1);
                pos = [rowIdx,colIdx];
                anchorIdx = bestAnchorIdx(t,1);

                mask(pos(1,1),pos(1,2),anchorIdx,batch) = 1;
                confMask(pos(1,1),pos(1,2),anchorIdx,batch) = 1;

                % Calculate the shift in ground-truth boxes.
                tShiftX = truthBatch(t,1)-pos(1,2)+1;
                tShiftY = truthBatch(t,2)-pos(1,1)+1;
                tShiftW = log(truthBatch(t,3)/anchors(anchorIdx,2));
                tShiftH = log(truthBatch(t,4)/anchors(anchorIdx,1));

                % Update the target box.
                tx(pos(1,1),pos(1,2),anchorIdx,batch) = tShiftX;
                ty(pos(1,1),pos(1,2),anchorIdx,batch) = tShiftY;
                tw(pos(1,1),pos(1,2),anchorIdx,batch) = tShiftW;
                th(pos(1,1),pos(1,2),anchorIdx,batch) = tShiftH;
                boxErrorScale(pos(1,1),pos(1,2),anchorIdx,batch) =
errorScale(t);
                tconf(rowIdx,colIdx,anchorIdx,batch) = 1;
                classIdx = (numClasses*(anchorIdx-1))+truthBatch(t,5);
                tclass(rowIdx,colIdx,classIdx,batch) = 1;
                classMask(rowIdx,colIdx,(numClasses*(anchorIdx-
1))+(1:numClasses),batch) = 1;
            end
        end
    end
    boxDeltaTarget(numPred,:) = [{tx} {ty} {tw} {th}];
    objectnessTarget{numPred,1} = tconf;
    classTarget{numPred,1} = tclass;
    maskTarget(numPred,:) = [{mask} {confMask} {classMask}];
    boxErrorScaleTarget{numPred,:} = boxErrorScale;
end
end

function iou =
getMaxIOUPredictedWithGroundTruth(predx,predy,predw,predh,truth)
% getMaxIOUPredictedWithGroundTruth computes the maximum intersection
over
%  union scores for every pair of predictions and ground-truth boxes.

[h,w,c,n] = size(predx);
iou = zeros([h w c n],'like',predx);
```

```
% For each batch prepare the predictions and ground-truth.
for batchSize = 1:n
    truthBatch = truth(:,1:4,1,batchSize);
    truthBatch = truthBatch(all(truthBatch,2),:);
    predxb = predx(:,:,:,batchSize);
    predyb = predy(:,:,:,batchSize);
    predwb = predw(:,:,:,batchSize);
    predhb = predh(:,:,:,batchSize);
    predb = [predxb(:),predyb(:),predwb(:),predhb(:)];

    % Convert from center xy coordinate to topleft xy coordinate.
    predb = [predb(:,1)-predb(:,3)./2, predb(:,2)-predb(:,4)./2,
predb(:,3), predb(:,4)];

    % Compute and extract the maximum IOU of predictions with ground-
truth.
    try
        overlap = bboxOverlapRatio(predb, truthBatch);
    catch me
        if(any(isnan(predb(:))))
            error(me.message + " NaN/Inf has been detected during
training. Try reducing the learning rate.");
        else
            error(me.message + " Invalid groundtruth. Check that your
ground truth boxes are not empty and finite, are fully contained within
the image boundary, and have positive width and height.");
        end
    end

    maxOverlap = max(overlap,[],2);
    iou(:,:,:,batchSize) = reshape(maxOverlap,h,w,c);
end
end

%------------------------ Model Gradients ----------------------------
function [gradients, state, info] = modelGradients(detector, XTrain,
YTrain, penaltyThreshold)
inputImageSize = size(XTrain,1:2);

% Gather the ground truths in the CPU for post processing
YTrain = gather(extractdata(YTrain));

% Extract the predictions from the detector.
[gatheredPredictions, YPredCell, state] = forward(detector, XTrain);

% Generate target for predictions from the ground truth data.
[boxTarget, objectnessTarget, classTarget, objectMaskTarget,
boxErrorScale] = generateTargets(gatheredPredictions,...
    YTrain, inputImageSize, detector.AnchorBoxes, penaltyThreshold);

% Compute the loss.
boxLoss = bboxOffsetLoss(YPredCell(:,[2 3 7
8]),boxTarget,objectMaskTarget,boxErrorScale);
objLoss =
objectnessLoss(YPredCell(:,1),objectnessTarget,objectMaskTarget);
clsLoss =
classConfidenceLoss(YPredCell(:,6),classTarget,objectMaskTarget);
totalLoss = boxLoss + objLoss + clsLoss;

info.boxLoss = boxLoss;
```

```
info.objLoss = objLoss;
info.clsLoss = clsLoss;
info.totalLoss = totalLoss;

% Compute gradients of learnables with regard to loss.
gradients = dlgradient(totalLoss, detector.Learnables);
end

%------------------------ Objectness loss ----------------------------
function objLoss = objectnessLoss(objectnessPredCell,
objectnessDeltaTarget, boxMaskTarget)
% Binary cross-entropy loss for objectness score.
objLoss = sum(cellfun(@(a,b,c)
crossentropy(a.*c,b.*c,'TargetCategories','independent'),objectnessPredC
ell,objectnessDeltaTarget,boxMaskTarget(:,2)));
end

%----------------- Piecewise learnrate and warmup ---------------------
function currentLR = piecewiseLearningRateWithWarmup(iteration, epoch,
learningRate, warmupPeriod, numEpochs)
% The piecewiseLearningRateWithWarmup function computes the current
% learning rate based on the iteration number.
persistent warmUpEpoch;

if iteration <= warmupPeriod
    % Increase the learning rate for number of iterations in warmup
period.
    currentLR = learningRate * ((iteration/warmupPeriod)^4);
    warmUpEpoch = epoch;
elseif iteration >= warmupPeriod && epoch <
warmUpEpoch+floor(0.6*(numEpochs-warmUpEpoch))
    % After warm up period, keep the learning rate constant if the
remaining number of epochs is less than 60 percent.
    currentLR = learningRate;

elseif epoch >= warmUpEpoch + floor(0.6*(numEpochs-warmUpEpoch)) &&
epoch < warmUpEpoch+floor(0.9*(numEpochs-warmUpEpoch))
    % If the remaining number of epochs is more than 60 percent but less
    % than 90 percent multiply the learning rate by 0.1.
    currentLR = learningRate*0.1;

else
    % If remaining epochs are more than 90 percent multiply the learning
    % rate by 0.01.
    currentLR = learningRate*0.01;
end
end

%--------------------------- Preprocess data ---------------------------
function data = preprocessData(data, targetSize)
% Resize the images and scale the pixels to between 0 and 1. Also scale
the
% corresponding bounding boxes.

for ii = 1:size(data,1)
    I = data{ii,1};
    imgSize = size(I);

    % Convert an input image with single channel to 3 channels.
    if numel(imgSize) < 3
```

```
        I = repmat(I,1,1,3);
    end
    bboxes = data{ii,2};

    I = im2single(imresize(I,targetSize(1:2)));
    scale = targetSize(1:2)./imgSize(1:2);
    bboxes = bboxresize(bboxes,scale);

    data(ii, 1:2) = {I, bboxes};
end
end

%--------------------------- Update plots ---------------------------
function updatePlots(lossPlotter, learningRatePlotter, iteration,
currentLR, totalLoss)
% Update loss and learning rate plots.
addpoints(lossPlotter, iteration,
double(extractdata(gather(totalLoss))));
addpoints(learningRatePlotter, iteration, currentLR);
drawnow
end

%----------------------- Validate input data ---------------------------
function validateInputData(ds)
% Validates the input images, bounding boxes and labels and displays the
% paths of invalid samples.

% Copyright 2021 The MathWorks, Inc.

% Path to images
info = ds.UnderlyingDatastores{1}.Files;

ds = transform(ds, @isValidDetectorData);
data = readall(ds);

validImgs = [data.validImgs];
validBoxes = [data.validBoxes];
validLabels = [data.validLabels];

msg = "";

if(any(~validImgs))
    imPaths = info(~validImgs);
    str = strjoin(imPaths, '\n');
    imErrMsg = sprintf("Input images must be non-empty and have 2 or 3
dimensions. The following images are invalid:\n") + str;
    msg = (imErrMsg + newline + newline);
end

if(any(~validLabels))
    imPaths = info(~validLabels);
    str = strjoin(imPaths, '\n');
    labelErrMsg = sprintf("Labels must be non-empty and categorical. The
following images have invalid labels:\n") + str;

    msg = (msg + labelErrMsg + newline);
end

if(~isempty(msg))
    error(msg);
end
```

```
end

function out = isValidDetectorData(data)
% Checks validity of images, bounding boxes and labels
for i = 1:size(data,1)
    I = data{i,1};
    boxes = data{i,2};
    labels = data{i,3};

    imageSize = size(I);
    mSize = size(boxes, 1);

    out.validImgs(i) = iCheckImages(I);
    out.validBoxes(i) = iCheckBoxes(boxes, imageSize);
    out.validLabels(i) = iCheckLabels(labels, mSize);
end

end

function valid = iCheckImages(I)
% Validates the input images.

valid = true;
if ndims(I) == 2
    nDims = 2;
else
    nDims = 3;
end
% Define image validation parameters.
classes        = {'numeric'};
attrs          = {'nonempty', 'nonsparse', 'nonnan', 'finite', 'ndims',
nDims};
try
    validateattributes(I, classes, attrs);
catch
    valid = false;
end
end

function valid = iCheckBoxes(boxes, imageSize)
% Validates the ground-truth bounding boxes to be non-empty and finite.

valid = true;
% Define bounding box validation parameters.
classes = {'numeric'};
attrs   = {'nonempty', 'integer', 'nonnan', 'finite', 'positive',
'nonzero', 'nonsparse', '2d', 'ncols', 4};
try
    validateattributes(boxes, classes, attrs);
    % Validate if bounding box in within image boundary.
    validateattributes(boxes(:,1)+boxes(:,3)-1, classes, {'<=',
imageSize(2)});
    validateattributes(boxes(:,2)+boxes(:,4)-1, classes, {'<=',
imageSize(1)});
catch
    valid = false;
end
end

function valid = iCheckLabels(labels, mSize)
```

```
% Validates the labels.

valid = true;
% Define label validation parameters.
classes = {'categorical'};
attrs   = {'nonempty', 'nonsparse', '2d', 'ncols', 1, 'nrows', mSize};
try
    validateattributes(labels, classes, attrs);
catch
    valid = false;
end
end
```

C. SSD code

```
clc; clear;

%---------------------------- LOAD TRAINING DATA  -----------------

myData = load('Pedestrians\train\trainLabels.mat');
names = myData.gTruth.DataSource.Source(:,1);
person =  myData.gTruth.LabelData.person;
Dataset = table(names, person);

% Shuffling train data
rng(0);
shuffledIdx = randperm(height(Dataset));
trainingData = Dataset(shuffledIdx,:);

% Image datastore
imds = imageDatastore(trainingData.names);

% label datastore
blds = boxLabelDatastore(trainingData(:,2:end));

% combine
ds = combine(imds,blds);

% Validating data
validateInputData(ds);

% Read combined data
trainData = read(ds);
I = trainData{1};
bb = trainData{2};
annImg = insertShape(I,'rectangle',bb);
annImg = imresize(annImg,2);
figure
imshow(annImg)

%Creating SSD Object Detection Network
imgInputSize = [300 300 3];

%Number of object classes to detect
classes = width(Dataset)-1;

%SSD layer
lgraph = ssdLayers(imgInputSize, classes, 'resnet50');
```

```
%Augment the training data
augmentedData = transform(ds,@augmentData);

%Preprocess the augmented training data to prepare for training
preprocessedData =
transform(augmentedData,@(data)preprocessData(data,imgInputSize));
trainData = read(preprocessedData);

% SSD Object Detector options
options = trainingOptions('sgdm', ...
    'MiniBatchSize', 8, ....
    'InitialLearnRate',0.001, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropPeriod', 30, ...
    'LearnRateDropFactor', 0.8, ...
    'MaxEpochs', 15, ...
    'VerboseFrequency', 50, ...
    'CheckpointPath', tempdir, ...
    'Shuffle','every-epoch');

  % Training  SSD.
  [ssdDetector, info] =
trainSSDObjectDetector(preprocessedData,lgraph,options);


%------------------------- TESTING AN IMG ----------------------------

myImg = imread('338.png');
myImg = imresize(myImg,imgInputSize(1:2));
[person,scores] = detect(ssdDetector,myImg, Threshold=0.3);
myImg = insertObjectAnnotation(myImg,'rectangle',person,scores);
figure
imshow(myImg)

%----------------- TESTING & EVALUATING METRICS ----------------------

% Load test data
myData2 = load('Pedestrians\test\testLabels.mat');
names = myData2.gTruth.DataSource.Source(:,1);
person =  myData2.gTruth.LabelData.person;
Dataset2 = table(names, person);

% Image datastore
imds2 = imageDatastore('Pedestrians\test');
% label datastore
blds2 = boxLabelDatastore(Dataset2(:,'person'));

% Run Trained Detector
testResults = detect(ssdDetector, imds2,Threshold=0.1);

% ----------------- Average Precision ----------------------------

[ap,recall,precision] = evaluateDetectionPrecision(testResults, blds2);

figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
```

```
title(sprintf('Average Precision = %.1f',ap))

% ------------------- Area Under Curve (AUC) -------------------------

x = recall;
y = precision;
auc = trapz(x,y);

% ---------------------- Log Average Miss Rate -------------------------

[am, fppi, missRate] = evaluateDetectionMissRate(testResults, blds2);

figure;
loglog(fppi, missRate);
grid on
title(sprintf('Log Average Miss Rate = %.2f', am))
```

**Note: The supporting functions are necessary for the main code to function.

Supporting functions for SSD obtained from MathWorks Website [35-37]:

```
% ------------------------- Augment data -----------------------------
function B = augmentData(A)
% Apply random horizontal flipping, and random X/Y scaling. Boxes that
get
% scaled outside the bounds are clipped if the overlap is above 0.25.
Also,
% jitter image color.
B = cell(size(A));
I = A{1};
sz = size(I);
if numel(sz)==3 && sz(3) == 3
    I = jitterColorHSV(I,...
        Contrast = 0.2,...
        Hue = 0,...
        Saturation = 0.1,...
        Brightness = 0.2);
end
% Randomly flip and scale image.
tform = randomAffine2d(XReflection = true, Scale = [1 1.1]);
rout = affineOutputView(sz,tform, BoundsStyle = 'CenterOutput');
B{1} = imwarp(I,tform,OutputView = rout);
% Sanitize boxes, if needed. This helper function is attached as a
% supporting file. Open the example in MATLAB to access this function.
A{2} = helperSanitizeBoxes(A{2});
% Apply same transform to boxes.
[B{2},indices] = bboxwarp(A{2},tform,rout,OverlapThreshold = 0.25);
B{3} = A{3}(indices);
% Return original data only when all boxes are removed by warping.
if isempty(indices)
    B = A;
end
end

% ------------------------- Augment data -----------------------------
%helperSanitizeBoxes Sanitize box data.
% This example helper is used to clean up invalid bounding box data.
Boxes with values <= 0 are removed.
```

```
%
% If none of the boxes are valid, this function passes the data through
to
% enable downstream processing to issue proper errors.
% Copyright 2020-2022 The Mathworks, Inc.
function boxes = helperSanitizeBoxes(boxes, ~)
persistent hasInvalidBoxes
valid = all(boxes > 0, 2);
if any(valid)
    if ~all(valid) && isempty(hasInvalidBoxes)
        % Issue one-time warning about removing invalid boxes.
        hasInvalidBoxes = true;
        warning('Removing ground truth bounding box data with values <=
0.')
    end
    boxes = boxes(valid,:);
end
end

% ------------------------- iSamePadding -----------------------------
function p = iSamePadding(FilterSize)
    p = floor(FilterSize / 2);
end

% ------------------------- Preprocess data ----------------------------
function data = preprocessData(data,targetSize)
% Resize image and bounding boxes to the targetSize.
sz = size(data{1},[1 2]);
scale = targetSize(1:2)./sz;
data{1} = imresize(data{1},targetSize(1:2));
% Sanitize boxes, if needed. This helper function is attached as a
% supporting file. Open the example in MATLAB to access this function.
data{2} = helperSanitizeBoxes(data{2});
% Resize boxes.
data{2} = bboxresize(data{2},scale);
end

% ---------------------- Validate input data ---------------------------
function validateInputData(ds)
% Validates the input images, bounding boxes and labels and displays the
% paths of invalid samples.

% Copyright 2021 The MathWorks, Inc.

% Path to images
info = ds.UnderlyingDatastores{1}.Files;

ds = transform(ds, @isValidDetectorData);
data = readall(ds);

validImgs = [data.validImgs];
validBoxes = [data.validBoxes];
validLabels = [data.validLabels];

msg = "";

if(any(~validImgs))
    imPaths = info(~validImgs);
    str = strjoin(imPaths, '\n');
```

```
    imErrMsg = sprintf("Input images must be non-empty and have 2 or 3
dimensions. The following images are invalid:\n") + str;
    msg = (imErrMsg + newline + newline);
end

if(any(~validBoxes))
    imPaths = info(~validBoxes);
    str = strjoin(imPaths, '\n');
    boxErrMsg = sprintf("Bounding box data must be M-by-4 matrices of
positive integer values. The following images have invalid bounding box
data:\n") ...
        + str;

    msg = (msg + boxErrMsg + newline + newline);
end

if(any(~validLabels))
    imPaths = info(~validLabels);
    str = strjoin(imPaths, '\n');
    labelErrMsg = sprintf("Labels must be non-empty and categorical. The
following images have invalid labels:\n") + str;

    msg = (msg + labelErrMsg + newline);
end

if(~isempty(msg))
    error(msg);
end
end

function out = isValidDetectorData(data)
% Checks validity of images, bounding boxes and labels
for i = 1:size(data,1)
    I = data{i,1};
    boxes = data{i,2};
    labels = data{i,3};

    imageSize = size(I);
    mSize = size(boxes, 1);

    out.validImgs(i) = iCheckImages(I);
    out.validBoxes(i) = iCheckBoxes(boxes, imageSize);
    out.validLabels(i) = iCheckLabels(labels, mSize);
end
end

function valid = iCheckImages(I)
% Validates the input images.

valid = true;
if ndims(I) == 2
    nDims = 2;
else
    nDims = 3;
end
% Define image validation parameters.
classes        = {'numeric'};
attrs          = {'nonempty', 'nonsparse', 'nonnan', 'finite', 'ndims',
nDims};
```

```
try
    validateattributes(I, classes, attrs);
catch
    valid = false;
end
end

function valid = iCheckBoxes(boxes, imageSize)
% Validates the ground-truth bounding boxes to be non-empty and finite.

valid = true;
% Define bounding box validation parameters.
classes = {'numeric'};
attrs   = {'nonempty', 'integer', 'nonnan', 'finite', 'positive',
'nonzero', 'nonsparse', '2d', 'ncols', 4};
try
    validateattributes(boxes, classes, attrs);
    % Validate if bounding box in within image boundary.
    validateattributes(boxes(:,1)+boxes(:,3)-1, classes, {'<=',
imageSize(2)});
    validateattributes(boxes(:,2)+boxes(:,4)-1, classes, {'<=',
imageSize(1)});
catch
    valid = false;
end
end

function valid = iCheckLabels(labels, mSize)
% Validates the labels.

valid = true;
% Define label validation parameters.
classes = {'categorical'};
attrs   = {'nonempty', 'nonsparse', '2d', 'ncols', 1, 'nrows', mSize};
try
    validateattributes(labels, classes, attrs);
catch
    valid = false;
end
end
```