Kier Yvanne Abot
BSCS 4A

Image Detection Algorithm Used:

Hog-SVM: HOG-SVM is a method for object detection that combines two key steps: the Histogram of Oriented Gradients (HOG) and Support Vector Machine (SVM). HOG works by examining an image to detect edges and patterns by looking at changes in brightness. It essentially maps out the shape of the object, such as the outline of a cat, by focusing on the directions in which brightness varies across different parts of the image. Once HOG has captured the general shape or pattern, SVM takes over to determine if this shape matches the object we want to detect. The SVM acts as a decision-making tool, deciding whether the pattern found is indeed what we are looking for, like a cat or another specified object. By combining HOG's pattern recognition with SVM's decision-making, HOG-SVM can accurately identify objects in various images.

Data Preparation is essential to prepare images uniformly, ensuring that variations in size, scale, and resolution do not impact the model's ability to learn. In the HOG-SVM approach, data preparation includes resizing images and normalizing them for consistency, as seen in the notebook, where images are resized to a fixed dimension (260x260) and converted to grayscale. For models requiring bounding boxes, such as SSD or YOLO, labeling objects in images is also necessary. In Model Building, the code employs the hog function from skimage.feature to extract HOG features, converting each image into a feature vector that represents its shape and texture. These features are then used to train an SVM classifier using sklearn.svm.

Accuracy: 59.91% – This indicates that approximately 60% of the model's predictions match the actual labels in the test set. While it's a moderate accuracy, it suggests there's room for improvement, particularly in refining the feature extraction process or optimizing the model further.

Precision: 65.74% – This shows the percentage of true positives out of all positive predictions made by the model. A precision of 65.74% implies that the model is relatively good at avoiding false positives, but there may still be misclassifications that could be fine-tuned, possibly by improving feature differentiation.

Recall: 59.91% – Matching the accuracy rate, this recall indicates the proportion of actual positives correctly identified by the model. With a recall of around 60%, the model is capturing the correct instances at a moderate rate but could benefit from enhanced feature extraction or additional data to better recognize all relevant objects.

Yolo – TensorFlow: YOLO-TensorFlow is a method for quickly detecting objects in images, using a model called YOLO (You Only Look Once) and built with the help of TensorFlow, a popular tool for machine learning. Unlike other methods that scan images in sections, YOLO looks at the whole image at once, which makes it much faster. It breaks down the image into a grid and predicts both the locations and types of objects in each part of the grid. For example, if it's given a photo of a street, YOLO can find and label objects like cars, people, and stop signs, all in one go. TensorFlow helps power YOLO by managing the complex

calculations that make this quick detection possible, allowing YOLO-TensorFlow to identify objects efficiently and in real-time.

Three paths are set up for the training, validation, and testing datasets (train_images_path, valid_images_path, and test_images_path). Each path points to a specific directory on Google Drive where the images are stored. The dataset is structured to ensure a proper flow for model training, validation, and testing. Organizing the images in separate folders for training, validation, and testing is a common approach in machine learning to evaluate model performance and avoid overfitting.

The labeling loop processes every image in each dataset split, creating the required .txt files for all images in the training, validation, and test datasets. Upon completion, the code prints a success message for each dataset split, confirming that label files have been generated for every image in the train, valid, and test folders.

Precision (0.0171) - Precision measures how accurately the model identifies only true positives (correct detections) without producing false positives (incorrect detections). A precision score of 0.0171 (or 1.71%) is very low, indicating that the model makes a large number of false-positive predictions, incorrectly identifying objects that aren't present. This low precision could mean the model may need further tuning or training on higher-quality data.

Recall (0.4981) - Recall indicates the proportion of actual positive cases (objects present in the images) that the model successfully detects. With a recall of 0.4981 (49.81%), the model can correctly identify almost half of all objects present, meaning it captures a fair number of objects but misses the other half. This recall rate suggests that the model has some capability for detection but may still overlook many relevant objects.

mAP@0.5 (Mean Average Precision at IoU threshold 0.5) - The mAP@0.5 score of 0.0447 (4.47%) measures the model's ability to locate objects with reasonable accuracy. It calculates the mean average precision across all classes using an Intersection over Union (IoU) threshold of 0.5, which considers predictions correct if they overlap with ground-truth boxes by at least 50%. An mAP@0.5 score of 4.47% is very low, suggesting that even when the model does detect objects, it may not place bounding boxes accurately enough.

Pytorch: PyTorch is a popular tool for building and training machine learning models, particularly suited for tasks like image and language understanding. It's designed to be flexible and user-friendly, allowing developers and researchers to easily experiment with various models and techniques. PyTorch enables users to work with "tensors," which function similarly to multi-dimensional arrays, efficiently storing and manipulating data. One of its standout features is dynamic computation, meaning that it constructs and modifies the model as it runs, simplifying testing and debugging. This feature, along with comprehensive support for automatic differentiation, makes PyTorch especially valuable for deep learning. PyTorch is primarily used for deep learning rather than traditional machine learning algorithms, as it provides tools to handle complex, layered neural networks and computations with ease, making it a go-to choose for developing and training sophisticated AI models.

The CustomDataset class, a subclass of torch.utils.data.Dataset, customized to load images and their associated labels for the object detection model. It takes in a root directory and applies specified

transformations (like resizing and normalization) to the images. The dataset is structured such that each class is represented by a subfolder within the main directory. CustomCNN, a convolutional neural network (CNN) class that includes convolutional, pooling, and fully connected layers. The architecture begins with a series of convolutional layers that progressively increase the number of channels, followed by pooling layers that reduce spatial dimensions. The final layers are fully connected (dense) layers that map the extracted features to the number of classes. The main training loop, including both training and validation phases. The training process spans 20 epochs, with each epoch involving a full pass through the training dataset. During training, the model is set to training mode, and the optimizer zeroes out gradients before each forward pass.

Accuracy (72.08%): The accuracy score reflects the proportion of correctly classified images in the test dataset. With an accuracy of 72.08%, the model correctly identifies objects in about 72 out of 100 test images. This is a moderate success rate, suggesting the model has learned to generalize reasonably well but may still misclassify some objects.

Precision (76.78%): Precision measures how accurately the model identifies true positives among all its positive predictions, specifically reflecting its ability to avoid false positives. A precision of 76.78% indicates that when the model predicts an object, it is correct approximately 77% of the time. This is a solid performance metric, showing that the model is fairly effective at minimizing incorrect detections.

Recall (72.08%): The recall score shows the model's ability to detect all actual positives in the dataset, representing the proportion of true positives identified out of all actual objects. With a recall of 72.08%, the model identifies around 72% of the objects in the images. The parity between recall and accuracy suggests that while the model is consistent, it may still miss some objects, leading to opportunities for improvement in detecting all relevant objects.

Average Inference Time (0.1476 seconds per image): This metric indicates the average time taken by the model to process each image and make predictions. An inference time of 0.1476 seconds per image translates to approximately 7 images per second. While not real-time, this speed is reasonable for tasks where immediate processing isn't required. For real-time or near-real-time applications, further optimizations could reduce inference time.

Summary of Model Suitability and Performance

- **Best Accuracy and Precision:** The custom CNN model in PyTorch achieves the best balance of accuracy (72.08%) and precision (76.78%), indicating it has learned well from the data and can generalize effectively.
- **Fastest Model for Real-Time Applications**: YOLO's speed (40.3 ms per image) makes it the most suitable for real-time tasks, though its low accuracy and precision suggest it needs further tuning to reach a reliable performance level.
- **Traditional Option**: HOG-SVM, while achieving a reasonable accuracy and recall, does not match the deep learning models' performance. It could serve as a baseline, but its slower inference and lower precision make it less ideal for complex or real-time tasks.