Evaluating Object Detection Models

**Table of Contents**

[Prepare the test data](#MW_H_A4EA5C73)  
[Run the detector on the test images](#MW_H_80E9B998)  
 [Out of memory errors](#MW_H_D7AA3F00)  
[Visualize detections](#MW_H_0590C75C)  
[Evaluate detection results](#MW_H_5A054AD4)  
[Visualize precision-recall curves for different classes](#MW_H_02F41042)  
[View the confusion matrix](#MW_H_7106)  
[Choose a detection threshold](#MW_H_020D2FA1)  
[Visualize mAP vs detection threshold curve](#MW_H_7D49718D)

In this script, you will evaluate an American Sign Language detector. The trained model and the code to train the model are provided in the course files.

Work through this script to evaluate the detector's performance on individual classes and as a whole.

**Note:** the detector provided with the course files is not the same as the detector used in the video, so you will see similar, but not identical, results. If you train a new model using the same settings we did, you will likely see different results. This is because the size of the dataset is relatively small, and there is some inherent randomness when training deep learning models. With only a few validation and test images, an extra missed detection or false positive will have a larger effect on the results than with a larger dataset.

# Prepare the test data

Run this section to load the trained detector and ground truth for the test images. Then create a combined datastore. For this example, no resizing or other pre-processing steps are required. If you had resized or preprocessed your images before training, you would need to apply those steps here.

% Load the detector and the ground truth for the test images

load aslTinyYolo4Detector.mat detector

load aslTestGT.mat aslTestGT

% Create a combined datastore of the images and bounding box data

[imds, blds] = objectDetectorTrainingData(aslTestGT);

testData = combine(imds, blds);

# **Run the detector on the test images**

The first step to evaluate your detector is to run the detector on your test images with a small confidence threshold. In the code below, we use 0.05. This means the detector is not picky and will return many detections, even if they have low confidence.

It took us about 10 seconds to run this on the 240 test images using a decently powered CPU. If you have a supported GPU, MATLAB will automatically use the GPU.

Examine the output table. Notice that the detector didn't detect any objects in some images and detected multiple objects in others.

testResults = detect(detector, testData, "Threshold", 0.05)

## Out of memory errors

When passing a datastore to the detect function a batch of images is pulled into memory for detection. Depending on the type of detector, image size, and your available memory, this could result in an out of memory error. If that is the case, use the "MiniBatchSize" Name-Value pair to reduce the number of images from the default of 128. An example is shown below.

**Important**: The detector is also run in the "Choose a detection threshold" section below. Make sure to update the call to the detect function to specify the MiniBatchSize if you are getting an out of memory error.

% testResults = detect(detector, testData, "Threshold", 0.05, "MiniBatchSize", 50)

# Visualize detections

Visualizing the detections on the images helps you evaluate the model performance. Select an image to view the detections (if any) and the associated scores.

numImages = height(testResults); % To set a limit on the control

imgIdx = 1;

img = readimage(imds, imgIdx);

label = string(testResults.Labels{imgIdx}) + " " + round(testResults.Scores{imgIdx},2);

img = insertObjectAnnotation(img, "rectangle", testResults.Boxes{imgIdx}, label);

imshow(img)

# Evaluate detection results

The evaluateObjectDetection function compares the detection results to the ground truth. The third input argument is the Intersection-over-Union (IoU) threshold used to define a true positive. The default value is 0.5. Here, we specify two IoU thresholds to show how a more strict location accuracy can impact the results.

The output is an object containing the evaluation results.

% Set the IoU threshold. The default value is 0.5

iou = [0.5 0.75];

% Evaluate the detector

metrics = evaluateObjectDetection(testResults, testData, iou);

% Get the class names to use later

classNames = metrics.ClassNames;

Use the summarize function to create tables summarizing the results. The mean average precision (mAP) with IoU = 0.5 is nearly 0.8 for this detector but decreases significantly with a more strict location accuracy requirement. The class summary output shows you how the detector behaves for each class. What letters have the biggest change in average precision between the two IoU thresholds?

[datasetSummary, classSummary] = summarize(metrics)

# Visualize precision-recall curves for different classes

Visualizing the precision-recall curves for individual classes gives you more detailed information on your detector. For example, the AP can be low due to missing many detections, many false positives, a few incorrect predictions with high confidence, or some combination these errors.

The precisionRecall function returns cell arrays for the precision and recall values already sorted by descending confidence score. Each row is a different object class and each column corresponds to an IoU threshold used when evaluating the detection results. The third output is a cell array of detection confidence scores for each class. For this example, the precision and recall outputs are 24x2 cell arrays since there are 24 classes and 2 IoU thresholds.

[precision, recall, scores] = precisionRecall(metrics);

Use the drop-down menu to select a class and view the precision-recall curves for the two IoU thresholds. View several letters, like A, E, C, G, O, and Y. What do the results tell you about the performance of the detector for different classes?

class = 1;

% Plot precision-recall for IoU 1

plot(recall{class,1}, precision{class,1}, "-o")

xlim([0 1])

xlabel("Recall")

ylabel("Precision")

title("Precision vs. Recall for " + classNames(class))

hold on

% Plot precision-recall for IoU 2

plot(recall{class,2}, precision{class,2}, "-o")

hold off

legend(["IoU = " + iou(1); "IoU = " + iou(2)], "Location", "southeast")

# View the confusion matrix

A confusion matrix helps you visualize the type of mistakes the models makes. Object detection models can leave objects unlabeled or assign labels to parts of the image with no ground truth objects. The label "unmatched" is used to identify true objects that were missed and detections that were not matched to any object.

The confusionMatrix function returns a cell array that has the confusion matrix for each IoU threshold. In this example, that means the first output is a 1x2 cell array. The first element of the cell array is the confusion matrix for IoU = 0.5 and the second element is the confusion matrix for IoU = 0.75. Use the confusionchart function to view the results.

Run this section to view the confusion matrix for the IoU = 0.5 results. Update the code to view the results of an IoU of 0.75.

[cm, confusionClasses] = confusionMatrix(metrics);

confusionchart(cm{1}, confusionClasses)

# Choose a detection threshold

When you use your detector in a real scenario, you need to choose an appropriate detection threshold. You could run the detector with a low threshold and write code to remove detections based on the individual class results. Here, we show how to find a global threshold that balances precision and recall across all classes.

To find an appropriate threshold for your detector:

1. Create a vector of threshold values to test
2. Run the detector with each threshold value
3. Evaluate the detector with the evaluateObjectDetection function
4. Use the summarize function to find the mAP
5. Plot mAP vs. detection threshold to determine where mAP sharply decreases

Run the code below to calculate mAP for a series of detection thresholds. The results use an IoU of 0.5.

**Note:** This may take a few minutes because the detector is being applied to every test image at multiple confidence thresholds.

% Vector of detection threshold values to test

thresholds = 0.1:0.1:0.9;

% Create a vector to store the results

mAPs = zeros(length(thresholds),1);

% Create a waitbar to monitor progress

f = waitbar(0);

% Loop through the detection thresholds, evaluate results, and save the AP

for ii = 1:length(thresholds)

results = detect(detector, testData, "Threshold", thresholds(ii));

currentMetrics = evaluateObjectDetection(results, testData, iou);

datasetSummary = summarize(currentMetrics);

mAPs(ii) = datasetSummary{:, "mAP0.5"};

waitbar(ii/length(thresholds), f, "Running detector and evaluating results")

end

close(f)

# Visualize mAP vs detection threshold curve

Ideally, the curve will be relatively flat as you increase the detection threshold. This means that false positives are being suppressed without decreasing recall much. However, increasing the confidence threshold too much will reduce recall, significantly lowering the mAP. In this example, choosing a detection threshold between 0.4 to 0.5 strikes a good balance.

plot(thresholds, mAPs, "-o")

xlabel("Detection Threshold")

ylabel("mAP")

title("mAP vs. Detection Threshold")

# 

*Copyright 2024 The MathWorks, Inc.*