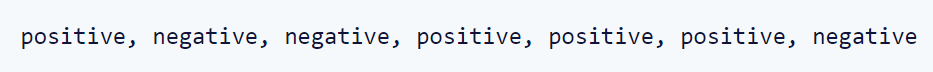
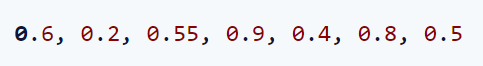
[Accuracy, Precision, and Recall in Deep Learning | Paperspace Blog](https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/)

Consider a binary classification problem having outputs positive and negative.

Assume below are the ground-truth labels:



Assume our model generates numbers (probabilities here):



In order to compare our predictions with the ground-truth labels, we need to convert predicted probabilities to labels positive and negative. In order to do this, we come up with some threshold value, say 0.5. So, all predicted probabilities >= 0.5 are mapped to positive, otherwise negative.

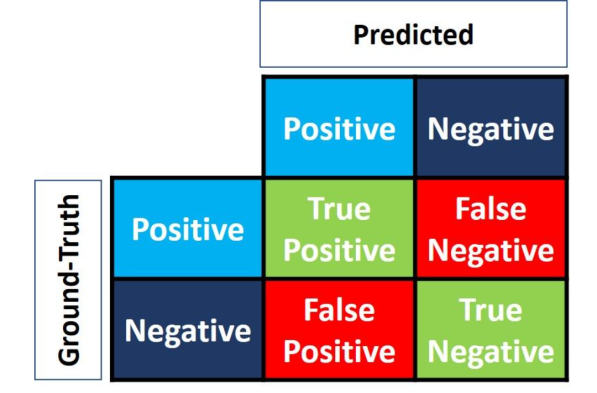
Now, the predicted labels become:



If you check the predicted and ground-truth labels, you will realize that there are 3 incorrect and 4 correct predictions.

**Confusion matrix:**

In order to evaluate a model, confusion matrix is used. It helps in understanding how ‘confused’ a model is in predicting correct class.

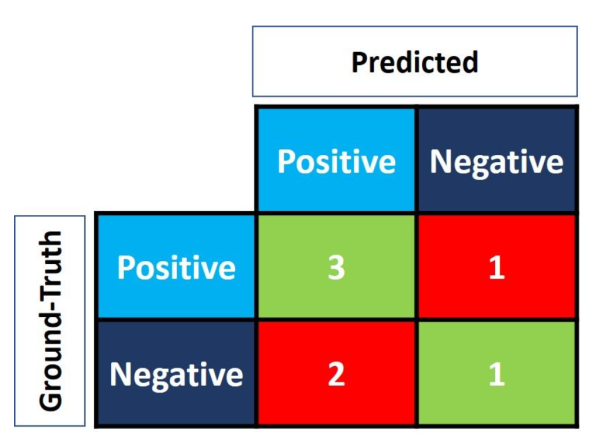


Note: “Ground-Truth” can be made columns and “Predicted” can be made rows

The four cells are named as below:

True means that the prediction and ground-truth match, and False means they don’t. Positive and Negative refers to the predicted label.

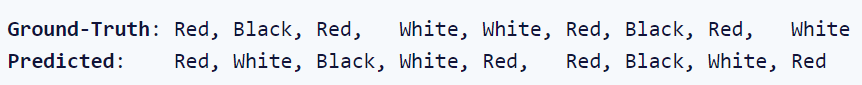
For the above binary classification example, we have:



We want to maximize the True positive and True Negative and minimize the False Positive and False Negative.

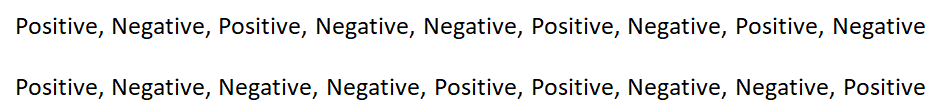
For multi-class problem:

Assume we have a 3-class problem. Below are the ground-truth and predicted labels:

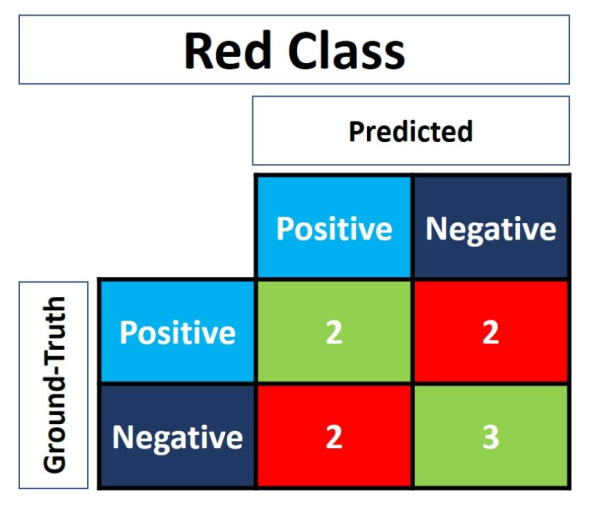


Confusion matrix can be calculated only when there are two classes. So, in case of multi-class classification, we need to compute confusion matrix per class separately. Since in the above example, we have 3 classes, we will have 3 confusion matrices.

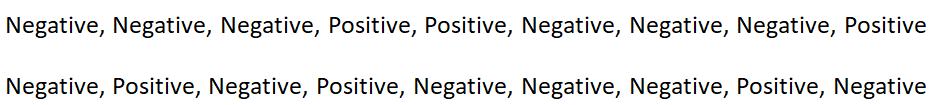
For Red class, the labels above become:

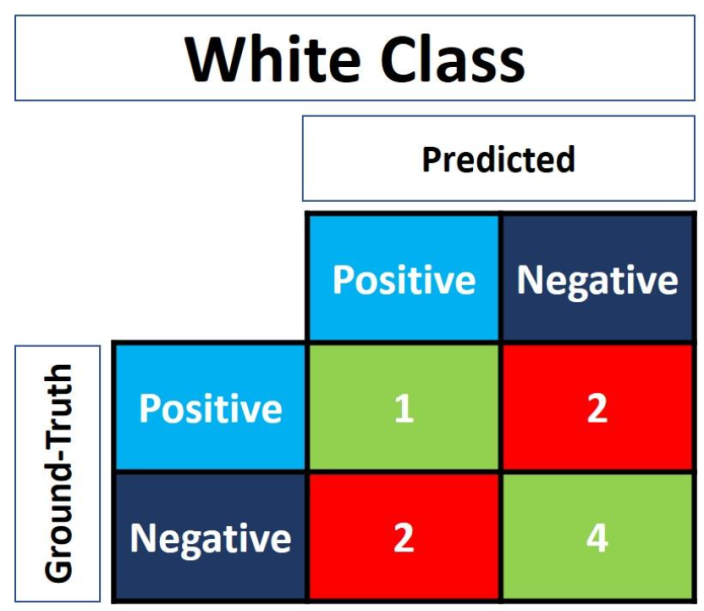


So, the confusion matrix is:

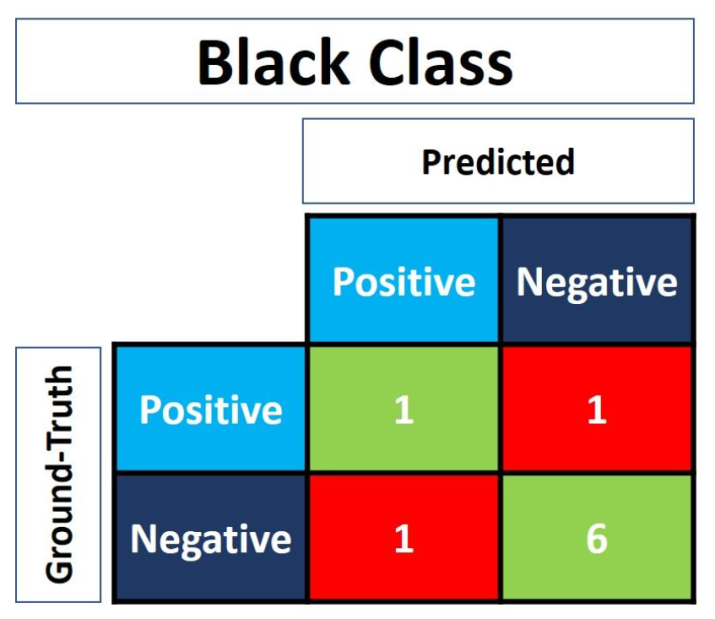


Similarly, for white class:





For black class:



Calculating confusion matrix using sklearn:

*y\_true = ["positive", "negative", "negative", "positive", "positive", "positive", "negative"]*

*y\_pred = ["positive", "negative", "positive", "positive", "negative", "positive", "positive"]*

*r = sklearn.metrics.confusion\_matrix(y\_true, y\_pred)*

*print(r)*

*array([[1, 2],*

*[1, 3]], dtype=int64)*

sklearn returns True Positive at the bottom-right corner and True Negative at the top-left corner.

We can flip the matrix:

*r = numpy.flip(r)*

*print(r)*

*array([[3, 1],*

*[2, 1]], dtype=int64)*

For multi-class problems:

*y\_true = ["Red", "Black", "Red", "White", "White", "Red", "Black", "Red", "White"]*

*y\_pred = ["Red", "White", "Black", "White", "Red", "Red", "Black", "White", "Red"]*

*r = sklearn.metrics.multilabel\_confusion\_matrix(y\_true, y\_pred, labels=["White", "Black", "Red"])*

*print(r)*

*array([*

*[[4 2]*

*[2 1]]*

*[[6 1]*

*[1 1]]*

*[[3 2]*

*[2 2]]], dtype=int64)*

The confusion matrices are returned in a single array in the order of the labels specified in the *labels* parameter.

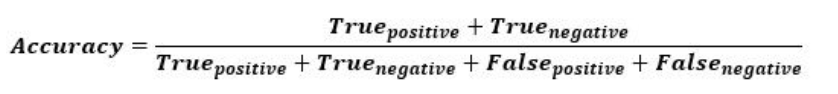
You can use numpy.flip() on individual confusion matrices.

**Accuracy, Precision, and Recall**

Using confusion matrix, more metrics can be calculated.

**Accuracy:**

It describes how a model performs across all classes. It is the ratio of the correct predictions to the total predictions.

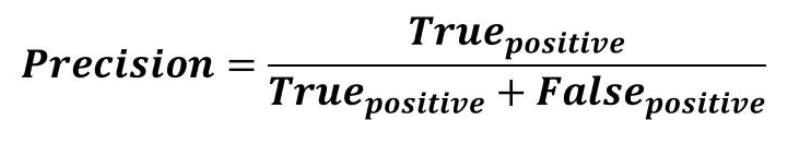


sklearn has a function called accuracy\_score() that can directly compute the accuracy.

*sklearn.metrics.accuracy\_score(y\_true, y\_pred)*

But accuracy can be deceptive when data is imbalanced; i.e. the no. of examples in some class(es) is much more/less than the no. of examples in the rest of the classes.

**Precision:**



Precision increases when the model makes many correct Positive classifications and/or fewer incorrect classifications.

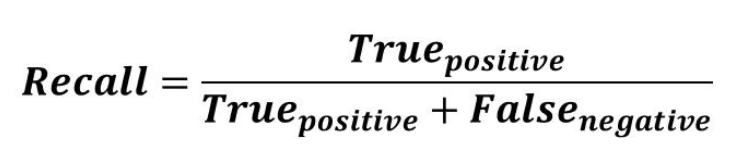
The precision helps to know how much the model is accurate when it says that a sample is Positive.

The goal of the precision is to classify all the Positive samples as Positive, and not misclassify a negative sample as Positive.

Using sklearn:

*precision = sklearn.metrics.precision\_score(y\_true, y\_pred, pos\_label="positive")*

**Recall:**

****

Recall cares only about how the positive samples are classified.

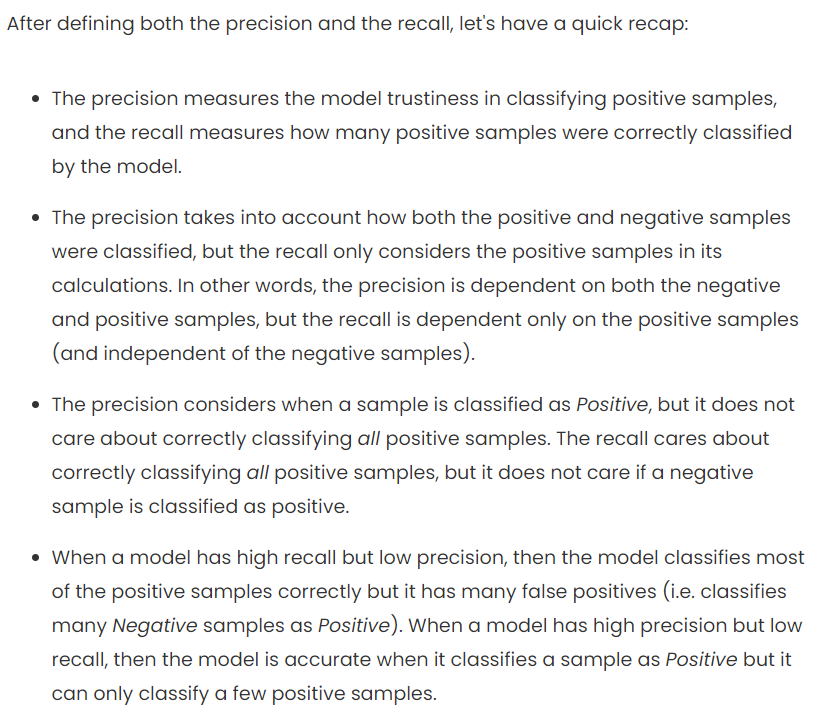
When a model classifies all the positive samples as Positive, the recall will be 100% even if all the negative samples were incorrectly classified as Positive.

When the recall is high, it means the model can classify all the positive samples correctly as Positive. Thus, the model can be trusted in its ability to detect positive samples.

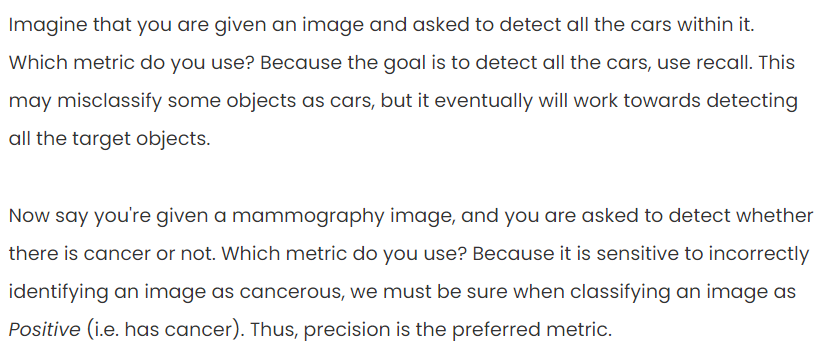
Using sklearn:

*recall = sklearn.metrics.recall\_score(y\_true, y\_pred, pos\_label="positive")*

Comparing Precision and Recall:



When to use Precision and Recall:



[Mean Average Precision (mAP) Explained | Paperspace Blog](https://blog.paperspace.com/mean-average-precision/)

To evaluate object detection models, mean Average Precision (mAP) is used. It compares the ground-truth bounding box to the detected box and returns a score.

Precision and Recall are used to calculate mAP.

From the previous section:

The higher the precision, the more confident the model is when it classifies a sample as Positive. The higher the recall, the more positive samples the model correctly classified as Positive.

When a model has high recall but low precision, then the model classifies most of the positive samples correctly but it has many false positives (i.e. classifies many Negative samples as Positive). When a model has high precision but low recall, then the model is accurate when it classifies a sample as Positive but it may classify only some of the positive samples

So, there is a trade-off between the two. This trade-off is given by the precision-recall curve.

Using this curve, you can decide the threshold used to classify objects such that both the metrics are maximized.

This curve is plotted by computing precision and recall for different threshold values.

Assume we have:

*Thresholds = [0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65]*

*Precisions = [0.5625, 0.5714285714285714, 0.5714285714285714, 0.6363636363636364, 0.7, 0.875, 0.875, 1.0, 1.0, 1.0]*

*Recalls = [1.0, 0.8888888888888888, 0.8888888888888888, 0.7777777777777778, 0.7777777777777778, 0.7777777777777778, 0.7777777777777778, 0.6666666666666666, 0.5555555555555556, 0.4444444444444444]*

Once you have precision and recall values for various thresholds, plot the curve.

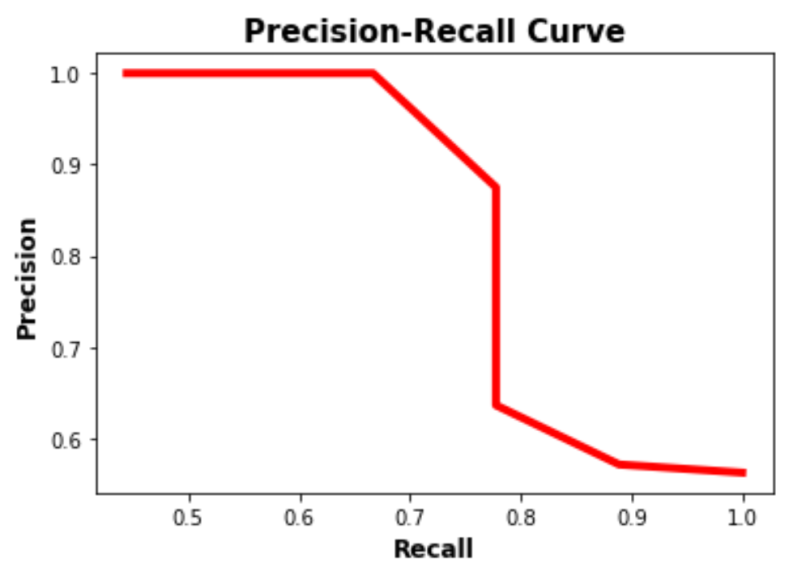
*matplotlib.pyplot.plot(recalls, precisions, linewidth=4, color="red")*

*matplotlib.pyplot.xlabel("Recall", fontsize=12, fontweight='bold')*

*matplotlib.pyplot.ylabel("Precision", fontsize=12, fontweight='bold')*

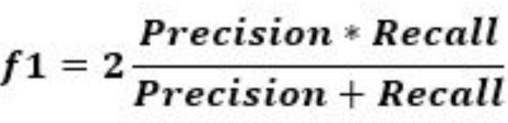
*matplotlib.pyplot.title("Precision-Recall Curve", fontsize=15, fontweight="bold")*

*matplotlib.pyplot.show()*



We can see that at (recall, precision) = (0.778, 0.875) both the metrics are high.

If the graph is complex, you can’t directly find such values. So, a better way is to use a metric called F1 score.



It measures the balance between precision and recall. When it is high it means both precision and recall are high.

If we calculate f1 scores for the above precision and recall pairs, we get below:

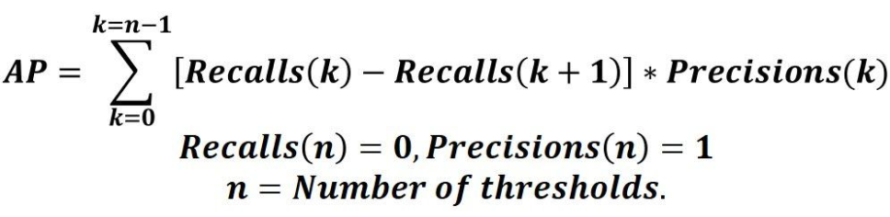
[0.72, 0.69565217, 0.69565217, 0.7, 0.73684211, 0.82352941, 0.82352941, 0.8, 0.71428571, 0.61538462]

The maximum value is 0.82352941, which is the 6th entry. So, we have best precision and recall as 0.875 and 0.778 respectively. The corresponding threshold is 0.45.

**Average Precision:**

A way to summarize the precision-recall curve into a single value representing is the average of all the precisions.

It is computed by subtracting two consecutive recall values and then multiplying the result by the precision corresponding to the first recall from these two consecutive recalls.



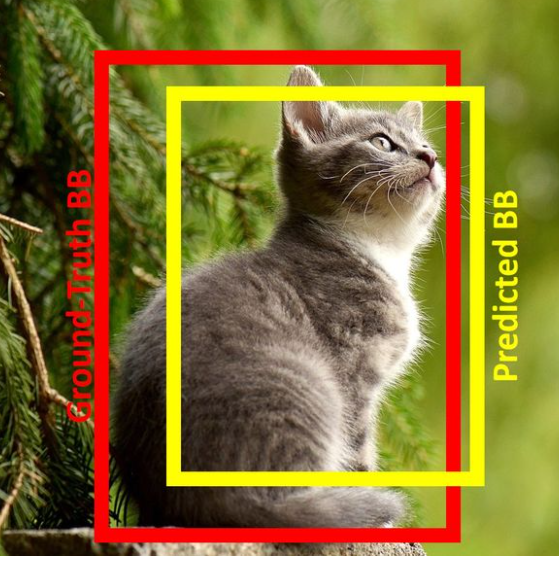
If precision and recall are numpy arrays, we have

*AP = numpy.sum((recalls[:-1] - recalls[1:]) \* precisions[:-1])*

In other words, the AP is the weighted sum of precisions at each threshold where the weight is the increase in recall.

**Intersection over Union (IoU):**

In object detection, a model predicts bounding boxes for the objects in the input image. In order to measure how good the predicted bounding box is, we define the IoU metric.





Assume each box is specifies by 4 numbers: x and y coordinates of the top-left corner and width and height of the box.

Using these values for the predicted box and the ground-truth box, find the intersection box. Then find the area of union. Finally compute IoU.

***def******intersection\_over\_union****(gt\_box, pred\_box):*

*inter\_box\_top\_left = [max(gt\_box[0], pred\_box[0]), max(gt\_box[1], pred\_box[1])]*

*inter\_box\_bottom\_right = [min(gt\_box[0]+gt\_box[2], pred\_box[0]+pred\_box[2]), min(gt\_box[1]+gt\_box[3], pred\_box[1]+pred\_box[3])]*

*inter\_box\_w = inter\_box\_bottom\_right[0] - inter\_box\_top\_left[0]*

*inter\_box\_h = inter\_box\_bottom\_right[1] - inter\_box\_top\_left[1]*

*intersection = inter\_box\_w \* inter\_box\_h*

*union = gt\_box[2] \* gt\_box[3] + pred\_box[2] \* pred\_box[3] - intersection*

*iou = intersection / union*

***return*** *iou, intersection, union*

To decide whether a model predicted the bounding box for an object well or not, you need to come up with a threshold value for IoU. If the computed IoU >= threshold, the model detected correctly.

**Mean Average Precision:**

Assume we have an image containing 10 objects of two classes.

To calculate mAP, calculate AP for each class. Then take the mean of the APs to get the mAP.

Assume for the first class, we have below data:

*y\_true = ["positive", "negative", "positive", "negative", "positive", "positive", "positive", "negative", "positive", "negative"]*

*pred\_scores = [0.7, 0.3, 0.5, 0.6, 0.55, 0.9, 0.75, 0.2, 0.8, 0.3]*

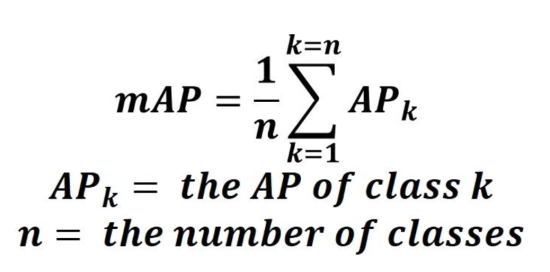
For the second class:

*y\_true = ["negative", "positive", "positive", "negative", "negative", "positive", "positive", "positive", "negative", "positive"]*

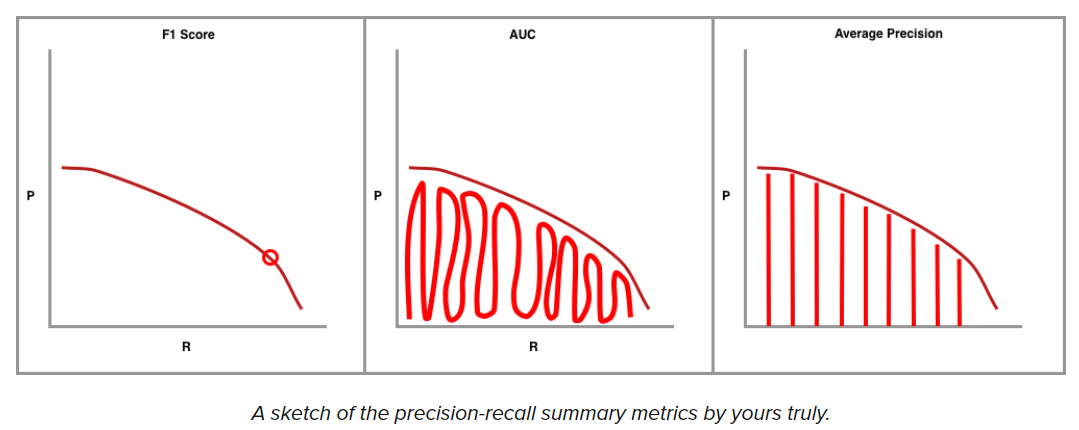
*pred\_scores = [0.32, 0.9, 0.5, 0.1, 0.25, 0.9, 0.55, 0.3, 0.35, 0.85]*

Calculate average precision (AP) for all the classes as explained in the previous sections.

Then compute mAP as:



[What is Mean Average Precision (mAP) in Object Detection? (roboflow.com)](https://blog.roboflow.com/mean-average-precision/)



The best F1 score gives the best precision-recall tradeoff and thus the best threshold to use for classification.

AUC finds all the area under the precision-recall curve.

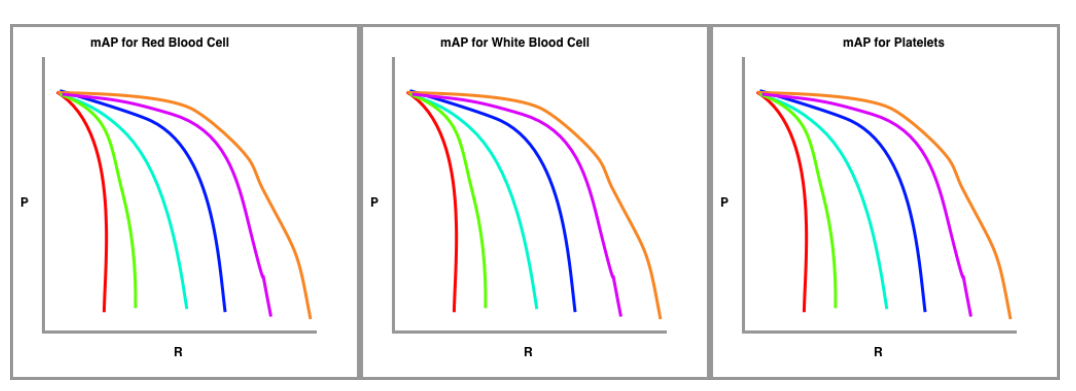
Average Precision is calculated as the weighted mean of average precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight.

The above techniques work for classification tasks. Now, lets apply these ideas to bounding box prediction.

To decide whether a predicted bounding box is correct or not, we use IoU.

Now, we plot the precision-recall curve using various IoU thresholds for each object class.

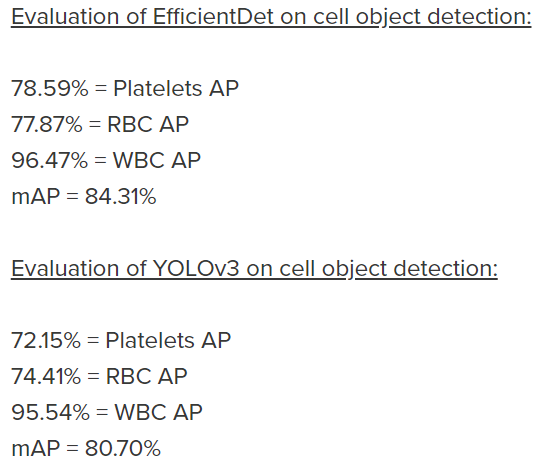
(Check next sections to understand how TP, FP, and FN are decided)



In the above example, we are trying to detect RBC, WBC, and platelet. So, for each class we plot precision-recall curve. We do this for various IOU thresholds to get multiple curves for each class.

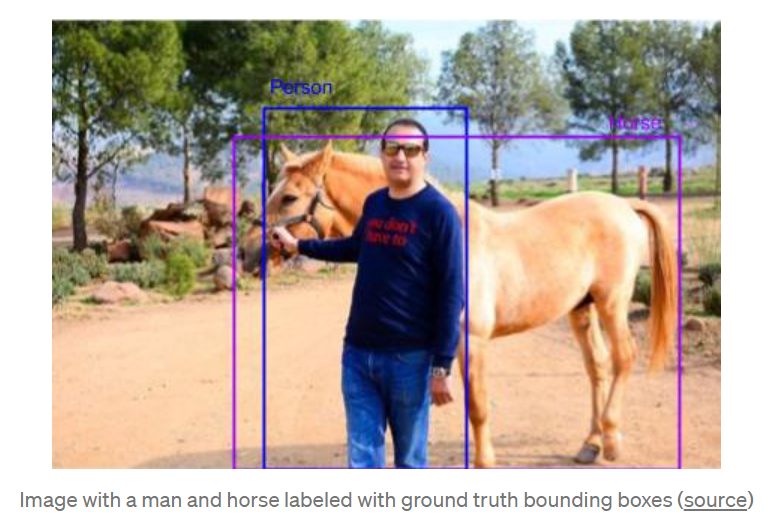
Now, we calculate Average Precision (AP) for each class individually. Finally, we take the mean of these APs to get mAP.

Example:

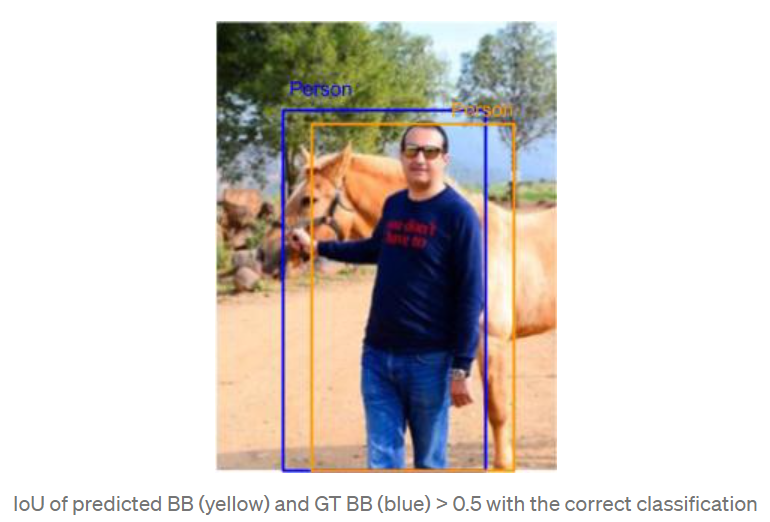


[Breaking Down Mean Average Precision (mAP) | by Ren Jie Tan | Towards Data Science](https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52#1a59)

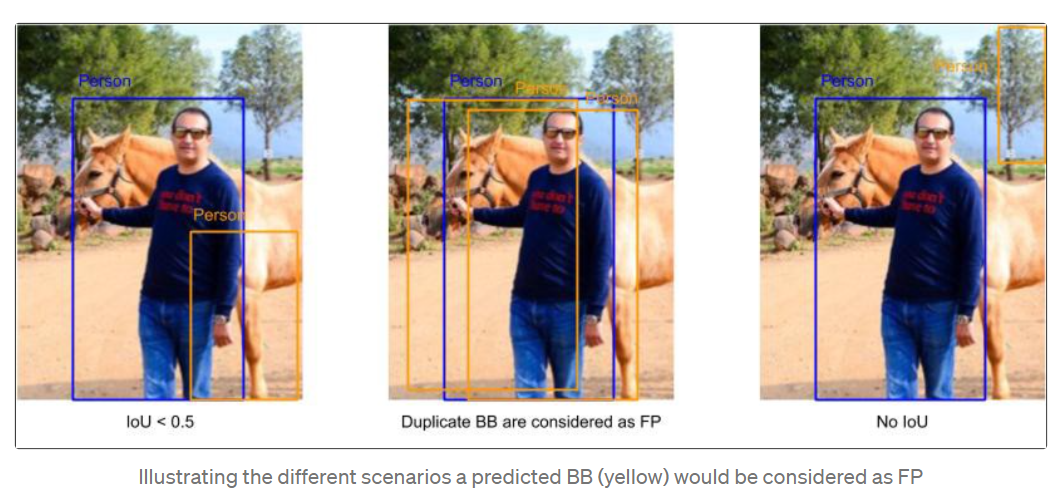
IoU is used to determine if a predicted bounding box is TP, FP, or FN. We don’t have TN as each image is assumed to have an object.



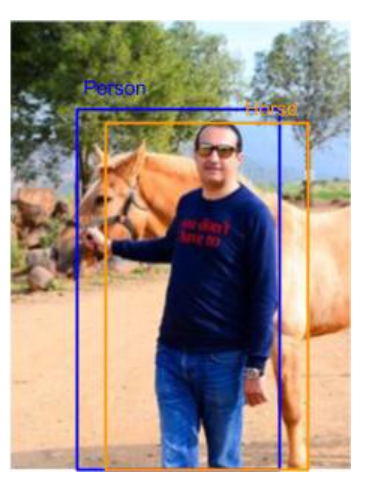
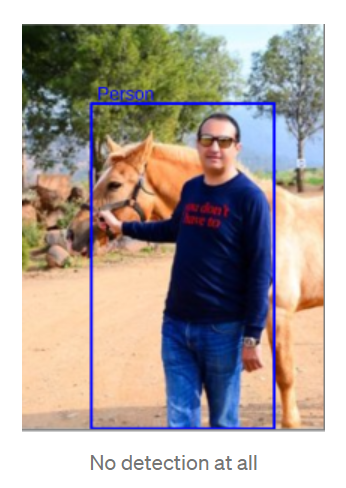
Let us ignore horse and consider only detecting the person. If the bounding box predicted for the person has IoU >= 0.5, we treat this as TP.



A bounding box prediction is considered FP if IoU < 0.5 or there are duplicate detections for the same object.



When the detector misses the object, we treat is as FN. There are two scenarios for this: when there is no detection at all and when the predicted bounding box has IoU>=0.5 but has the wrong class.



Once we have identified TP, FP, and FN examples, we calculate precision and recall for each object class.

SKIPPED rest as it became confusing. Check the next link.

[How To Calculate the mean Average Precision (mAP) - an overview | hungsblog](https://hungsblog.de/en/technology/how-to-calculate-mean-average-precision-map/)

For each prediction, we also get a confidence score (I think, this is the product of the probability that there is an object and the probability that the object is of the class under consideration.

We sort the predictions using these confidence scores and find out TP, FP, and FN. Then we calculate precision and recall.

Now, we want a single metric to specify how good a model is. AP does this.

Often, to compute AP, interpolation is used.

SKIPPED interpolation part

Now, mAP is computed by taking mean of APs for all the classes.

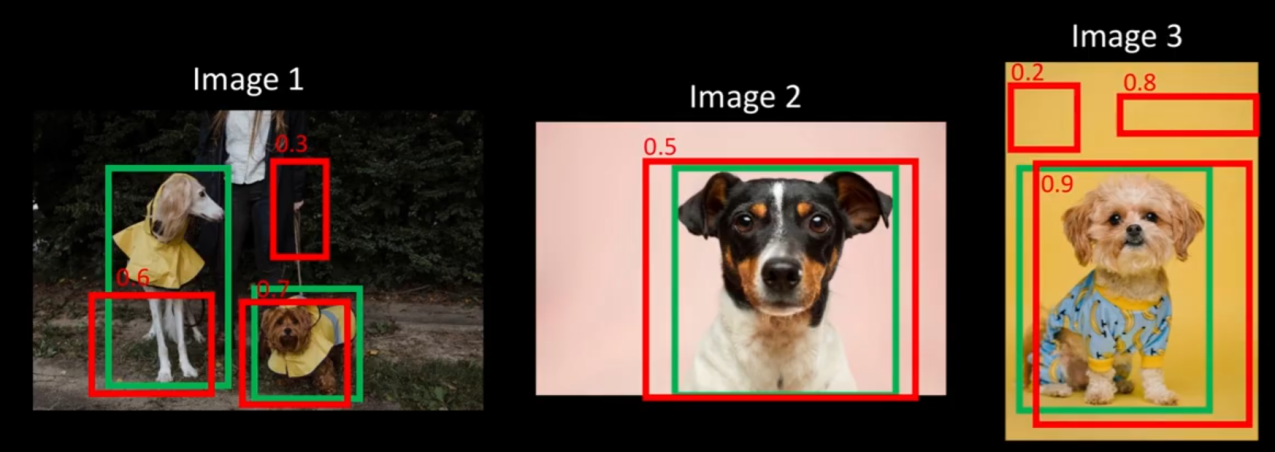
In Pascal VOC, mAP is computed using IoU threshold 0.5

In COCO challenge, mAP is computed by using various IoU thresholds.

[Mean Average Precision – Aladdin Persson video](https://www.youtube.com/watch?v=FppOzcDvaDI)

([mAP.mp4](../../../../../GitHub-Supplement/Computer%20Vision/Papers/object-detection/Mean%20Average%20Precision%20(mAP)%20Explained%20and%20PyTorch%20Implementation.mp4))

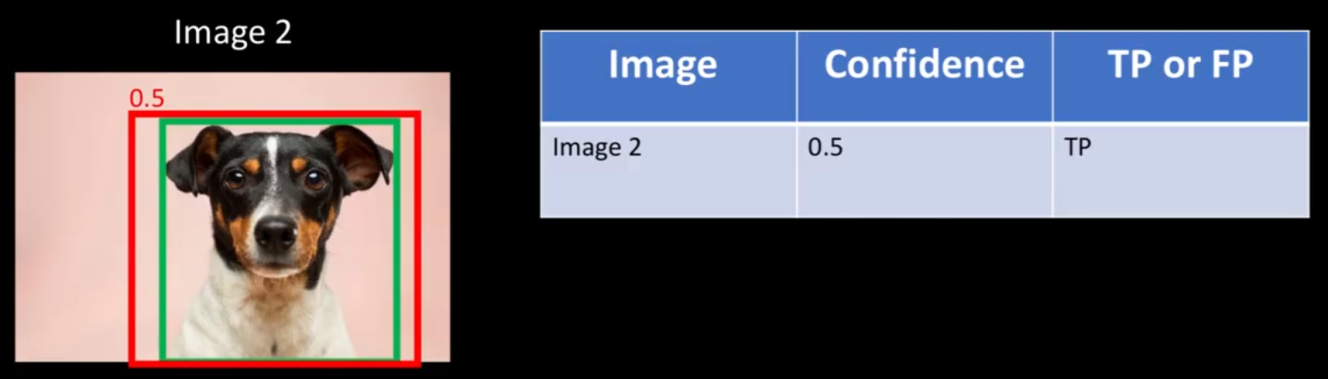
* Predict bounding boxes using your model

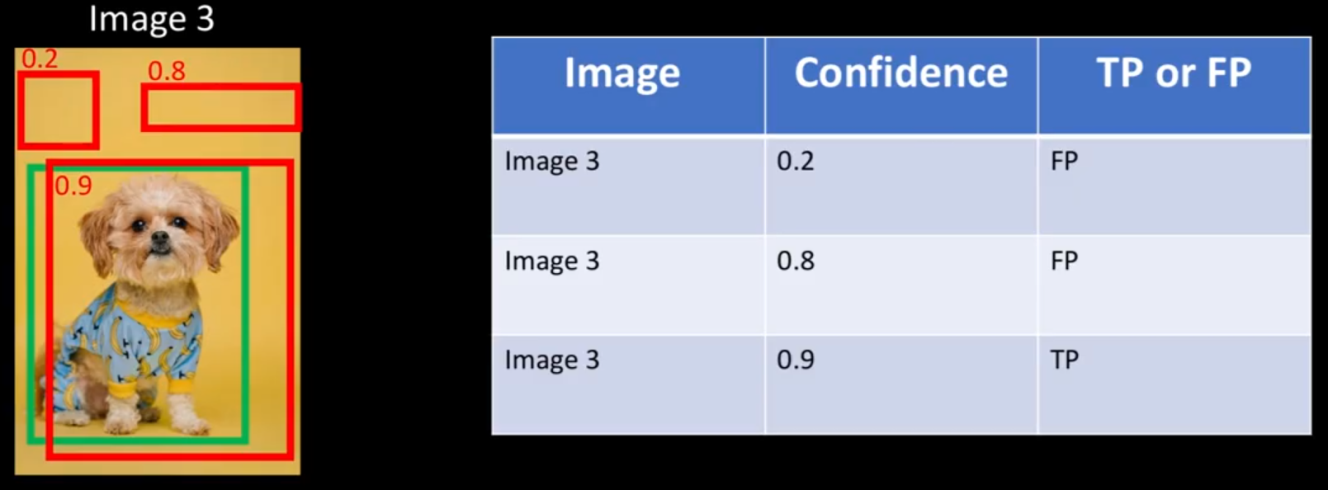


Green bounding boxes are predicted and the red ones are the expected ones.

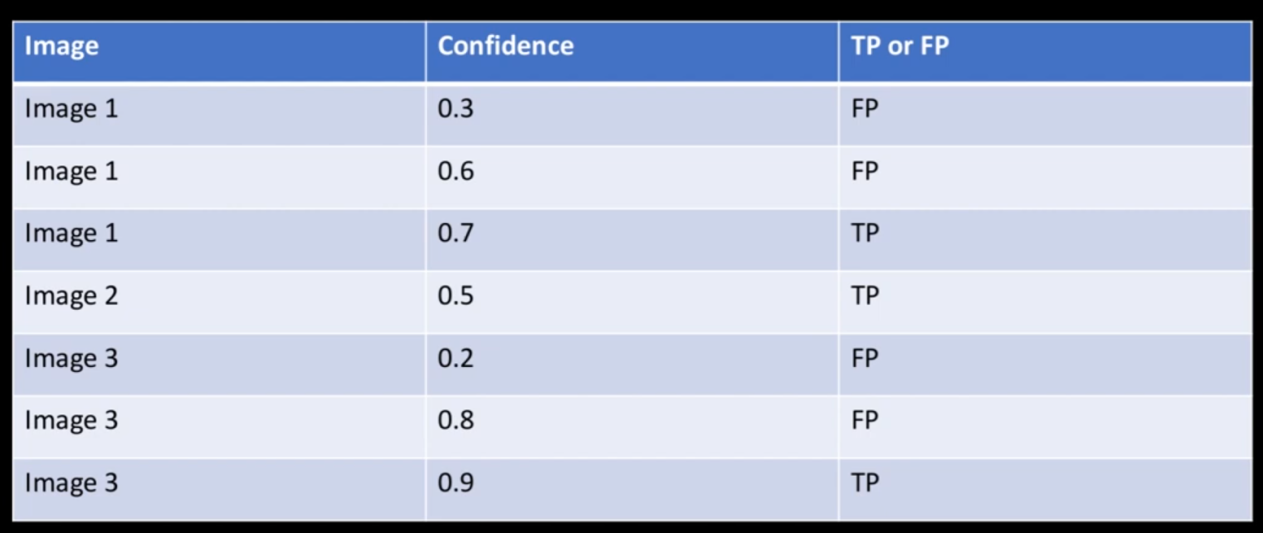
The number on each box is the probability that the model thinks that there is an object in the box.

Now, we use IOU to decide whether a box is TP, FP, or FN. 

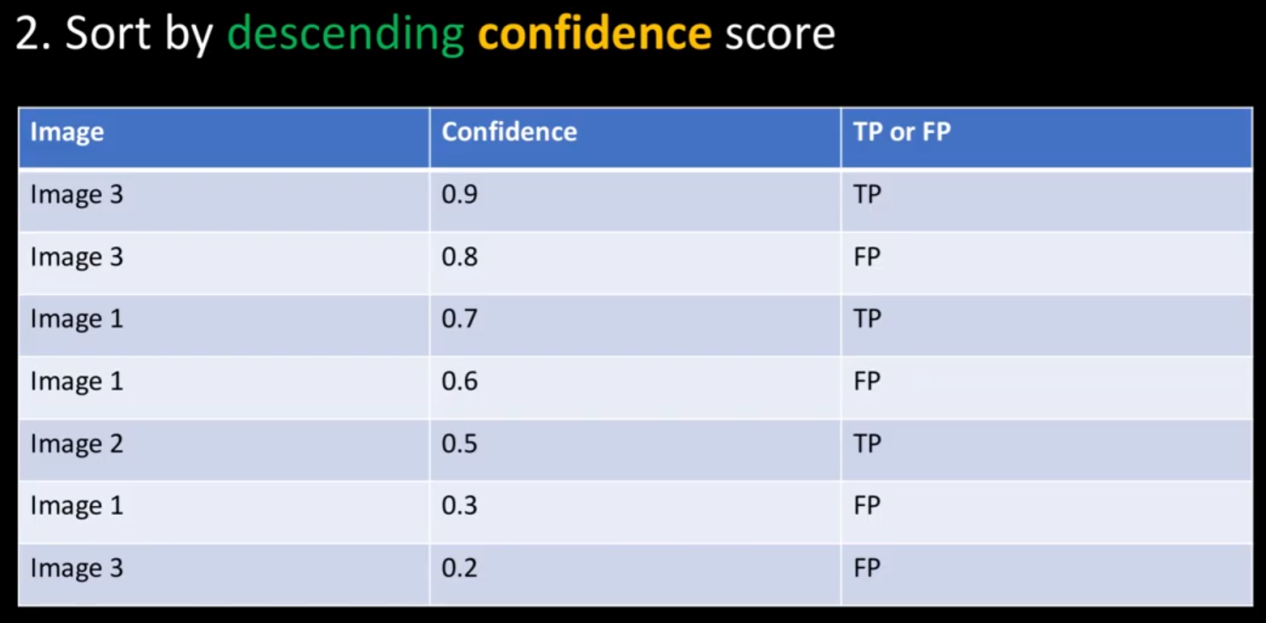




To summarize all the predicted bounding boxes:



* Sort the bounding boxes on the probability values



TP – the bounding box predicting the correct class and having IOU>=0.5

FP – the bounding box predicting wrong class and/or having IOU<0.5

(Note, if there are multiple bounding boxes predicting the same class and having IOU>=0.5, the one with the maximum IOU is chosen as TP and others are considered FP

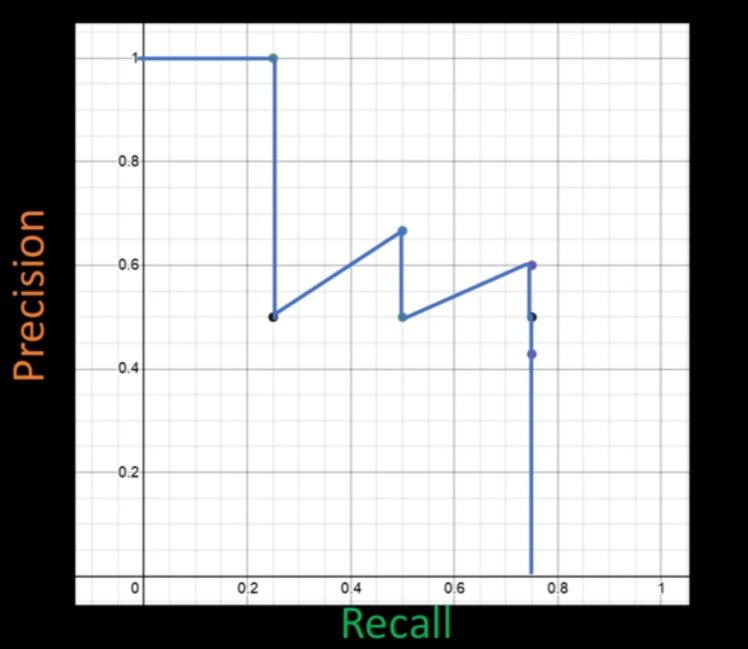
FN – the model doesn’t predict any bounding box even though it is expected

*TN – the model doesn’t predict bounding box, and there is no expected box; this condition is not applicable to object detection systems.*

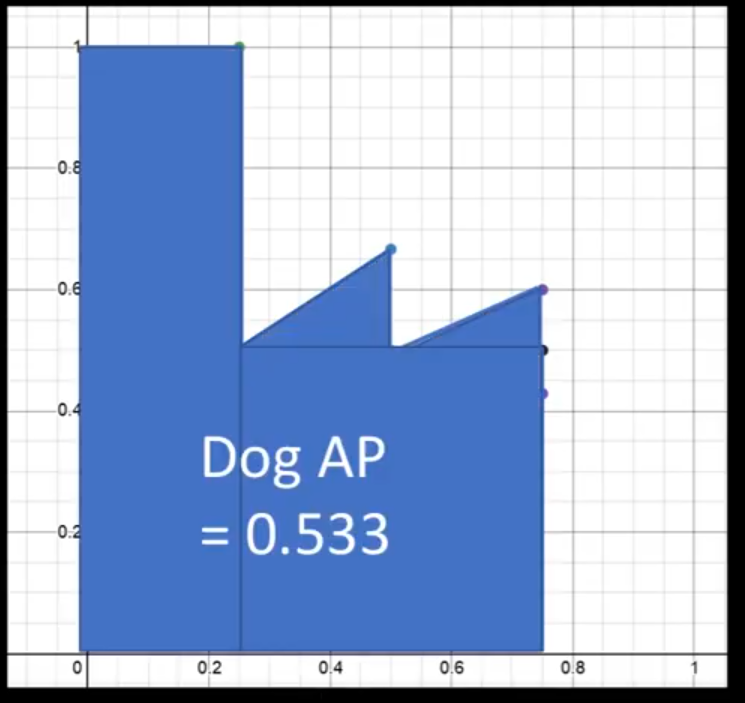
* Calculate running precision and recall as we go through all the model outputs



* Plot the Precision-Recall curve using the above table



* Average precision is basically the area under the precision-recall graph



So, for the dog class, we have AP=0.533

* Calculate AP for other classes.

(assuming we have cat and dog classes)

Cat AP: 0.74

Dog AP: 0.533

Mean Average Precision is the average of all APs.

mAP = (0.533 + 0.74)/2 = 0.6365

* All the above steps were done for IOU threshold=0.5. We need to do this for other thresholds as well – 0.5, 0.55, 0.6, …, 0.95

Taking average of all such mAPs gives us the final mAP value.