Object Detection and Tracking Using Deep Learning

Majeed Hussain , Akshay Kumar
Supervisor: Dr.David Fofi
University of Bourgogne
MSCV

I. Objective

The Objective of this Project is to build a model which detects objects and tracks them using Deep learning models and traditional Trackers.

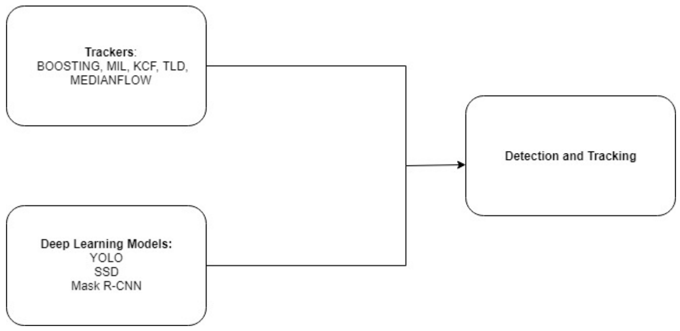
II. Procedure

We started out this Project from using trackers available in OpenCV library and Pre-trained Models for detection of Objects and later Combine them to do both tasks i.e, Detection and tracking at the same time.

The tasks divided into three parts they are:

1) Tracking Using BOOSTING, MIL, KCF, TLD, MEDIANFLOW.
2) Detection and Recognition using YOLO, SSD, Mask R-CNN.
3) Combining by using any one of the Tracker with deep learning Model to do both tasks at a same time.

The pipeline of the Project goes as follows:



A. Trackers

We used few trackers from OpenCV library they are BOOSTING, MIL, KCF, TLD ,MEDIANFLOW.

1) BOOSTING: This tracker is based on an online version of AdaBoost the algorithm that the HAAR cascade based face detector uses internally. This classifier needs to be trained at runtime with positive and negative examples of the object. The initial bounding box supplied by the user (or by another object detection algorithm) is taken as the positive example for the object, and many image patches outside the bounding box are treated as the background. Given a new frame, the classifier is run on every pixel in the neighborhood of the previous location and the score of the classifier is recorded. The new location of the object is the one where the score is maximum. So now we have one more positive example for the classifier. As more frames come in, the classifier is updated with this additional data.

Pros : None. This algorithm is a decade old and works ok, but I could not find a good reason to use it especially when other advanced trackers (MIL, KCF) based on similar principles are available.

Cons : Tracking performance is mediocre. It does not reliably know when tracking has failed.

2) MIL Tracker: This tracker is similar in idea to the BOOSTING tracker described above. The big difference is that instead of considering only the current location of the object as a positive example, it looks in a small neighborhood around the current location to generate several potential positive examples. You may be thinking that it is a bad idea because in most of these positive examples the object is not centered.

This is where Multiple Instance Learning (MIL) comes to rescue. In MIL, you do not specify positive and negative examples, but positive and negative bags. The collection of images in the positive bag are not all positive examples. Instead, only one image in the positive bag needs to be a positive example! In our example, a positive bag contains the patch centered on the current location of the object and also patches in a small neighborhood around it. Even if the current location of the tracked object is not accurate, when samples from the neighborhood of the current location are put in the positive bag, there is a good chance that this bag contains at least one image in which the object is nicely centered. MIL project page has more information for people who like to dig deeper into the inner workings of the MIL tracker.

Pros : The performance is pretty good. It does not drift as much as the BOOSTING tracker and it does a reasonable job under partial occlusion. If you are using OpenCV 3.0, this might be the best tracker available to you. But if you are using a higher version, consider KCF.

Cons : Tracking failure is not reported reliably. Does not recover from full occlusion.

3) KCF Tracker: KFC stands for Kernelized Correlation Filters. This tracker builds on the ideas presented in the previous two trackers. This tracker utilizes that fact that the multiple positive samples used in the MIL tracker have large overlapping regions. This overlapping data leads to some nice mathematical properties that is exploited by this tracker to make tracking faster and more accurate at the same time.

Pros: Accuracy and speed are both better than MIL and it reports tracking failure better than BOOSTING and MIL.

Cons : Does not recover from full occlusion.

4) TLD Tracker: TLD stands for Tracking, learning and detection. As the name suggests, this tracker decomposes the long term tracking task into three components (short term) tracking, learning, and detection. From the authors paper, The tracker follows the object from frame to frame. The detector localizes all appearances that have been observed so far and corrects the tracker if necessary. The learning estimates detectors errors and updates it to avoid these errors in the future. This output of this tracker tends to jump around a bit. For example, if you are tracking a pedestrian and there are other pedestrians in the scene, this tracker can sometimes temporarily track a different pedestrian than the one you intended to track. On the positive side, this track appears to track an object over a larger scale, motion, and occlusion. If you have a video sequence where the object is hidden behind another object, this tracker may be a good choice.

Pros : Works the best under occlusion over multiple frames. Also, tracks best over scale changes.

Cons : Lots of false positives making it almost unusable.

5) MEDIANFLOW Tracker: Internally, this tracker tracks the object in both forward and backward directions in time and measures the discrepancies between these two trajectories. Minimizing this ForwardBackward error enables them to reliably detect tracking failures and select reliable trajectories in video sequences.

In my tests, I found this tracker works best when the motion is predictable and small. Unlike, other trackers that keep going even when the tracking has clearly failed, this tracker knows when the tracking has failed.

Pros : Excellent tracking failure reporting. Works very well when the motion is predictable and there is no occlusion.

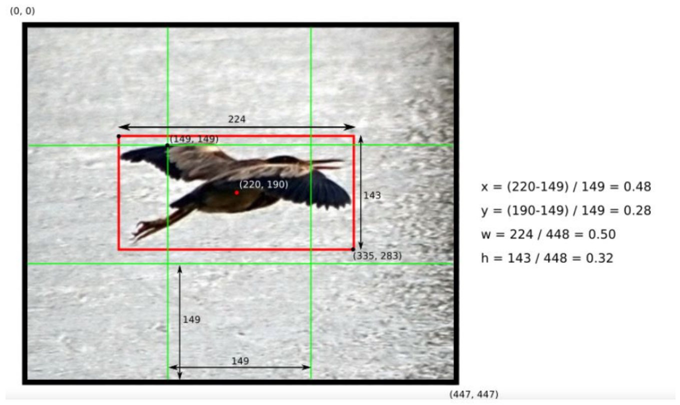
Cons : Fails under large motion.

B. Deep Learning Models

We used three models namely YOLOM SSD, Mask R-CNN for detection and recognition. Once we perform Detection we use trackers along with detection results and perform simultaneous detectio and tracking.

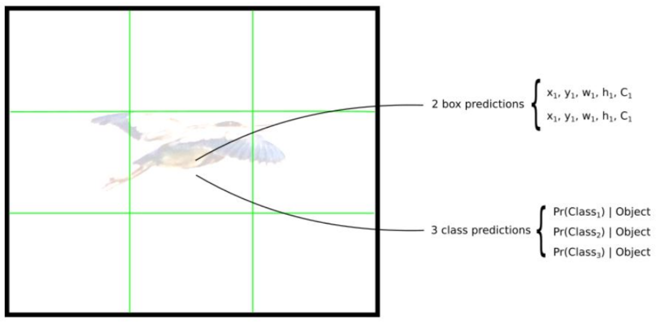
1) YOLO: YOLO stands for You Only Look Once, it is a network for object detection. The object detection task consists in determining the location on the image where certain objects are present, as well as classifying those objects. Previous methods for this, like R-CNN and its variations, used a pipeline to perform this task in multiple steps. This can be slow to run and also hard to optimize, because each individual component must be trained separately. YOLO does all tasks with a single neural network.So, to put it simple, you take an image as input, pass it through a neural network that looks similar to a normal CNN, and you get a vector of bounding boxes and class predictions in the output.It is based on regression instead of selecting interesting parts of an image, were predicting classes and bounding boxes for the whole image in one run of the algorithm.The first step to understanding YOLO is how it encodes its output. The input image is divided into an S x S grid of cells. For each object that is present on the image, one grid cell is said to be responsible for predicting it. That is the cell where the center of the object falls into.

Each grid cell predicts B bounding boxes as well as C class probabilities. The bounding box prediction has 5 components: (x, y, w, h, confidence). The (x, y) coordinates represent the center of the box, relative to the grid cell location (remember that, if the center of the box does not fall inside the grid cell, than this cell is not responsible for it). These coordinates are normalized to fall between 0 and 1. The (w, h) box dimensions are also normalized to [0, 1], relative to the image size. Lets look at an example:



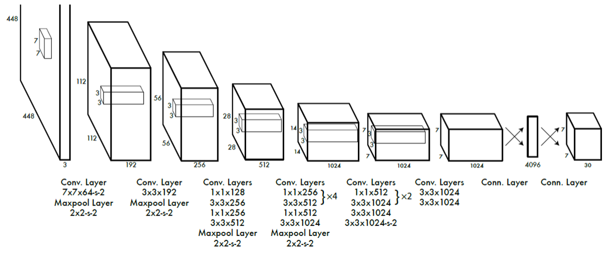
There is still one more component in the bounding box prediction, which is the confidence score.Formally we define confidence as Pr(Object) * IOU(pred, truth) . If no object exists in that cell, the confidence score should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.Note that the confidence reflects the presence or absence of an object of any class. In case you don't know what IOU is, take a look here.Now that we understand the 5 components of the box prediction, remember that each grid cell makes B of those predictions, so there are in total S x S x B * 5 outputs related to bounding box predictions.

It is also necessary to predict the class probabilities, Pr(Class(i) — Object). This probability is conditioned on the grid cell containing one object (see this if you dont know that conditional probability means). In practice, it means that if no object is present on the grid cell, the loss function will not penalize it for a wrong class prediction, as we will see later. The network only predicts one set of class probabilities per cell, regardless of the number of boxes B. That makes S x S x C class probabilities in total.Adding the class predictions to the output vector, we get a S x S x (B * 5 +C) tensor as output.



Each grid cell makes B bounding box predictions and C class predictions (S=3, B=2 and C=3 in this example)

2) Network: The network structure looks like a normal CNN, with convolutional and max pooling layers, followed by 2 fully connected layers in the end


448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

The architecture was crafted for use in the Pascal VOC dataset, where the authors used S=7, B=2 and C=20. This explains why the final feature maps are 7x7, and also explains the size of the output (7x7x(2*5+20)). Use of this network with a different grid size or different number of classes might require tuning of the layer dimensions.The sequences of 1x1 reduction layers and 3x3 convolutional layers were inspired by the GoogLeNet model.The final layer uses a linear activation function. All other layers use a leaky RELU.

3) Loss Function: The first part of the loss function is as follows:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 \qquad (1)$$

This equation computes the loss related to the predicted bounding box position (x,y). Dont worry about for now, just consider it a given constant. The function computes a sum over each bounding box predictor (j = 0.. B) of each grid cell. obj is defined as follows:

1) 1, If an object is present in grid cell i and the jth bounding box predictor is responsible for that prediction.
2) 0, otherwise

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be responsible for predicting an object based on which prediction has the highest current IOU with the ground truth.

The Second Part of loss function is as follows:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \qquad (2)$$

This is the loss related to the predicted box width / height.Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.

The third Part of loss function is as follows:

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2 \qquad (3)$$

Here we compute the loss associated with the confidence score for each bounding box predictor. C is the confidence score and is the intersection over union of the predicted bounding box with the ground truth. obj is equal to one when there is an object in the cell, and 0 otherwise. noobj is the opposite.The parameters that appear here and also in the first part are used to differently weight parts of the loss functions. This is necessary to increase model stability. The highest penalty is for coordinate predictions (coord = 5) and the lowest for confidence predictions when no object is present (noobj = 0.5).

The las part of loss function is as follows:

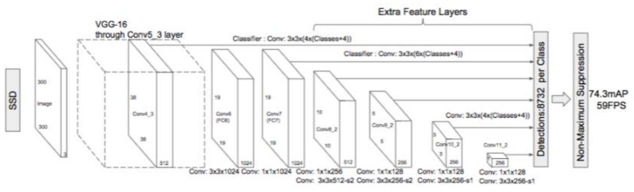
$$\sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2 \qquad (4)$$

It looks similar to a normal sum-squared error for classification, except for the obj term. This term is used because so we dont penalize classification error when no object is present on the cell.

4) SSD: SSD stands for Single shot Detector.

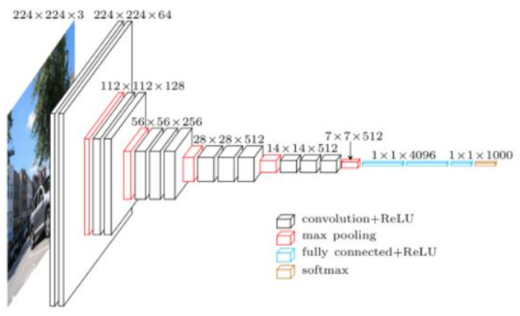
1) Single Shot: this means that the tasks of object localization and classification are done in a single forward pass of the network
2) MultiBox: this is the name of a technique for bounding box regression developed by Szegedy et al.
3) Detector: The network is an object detector that also classifies those detected objects.

Architecture:



Architecture of Single Shot MultiBox detector (input is 300x300x3)

As you can see from the diagram above, SSDs architecture builds on the venerable VGG-16 architecture, but discards the fully connected layers. The reason VGG-16 was used as the base network is because of its strong performance in high quality image classification tasks and its popularity for problems where transfer learning helps in improving results. Instead of the original VGG fully connected layers, a set of auxiliary convolutional layers from conv6 onwards were added, thus enabling to extract features at multiple scales and progressively decrease the size of the input to each subsequent layer.



VGG architecture (input is 224x224x3)

MultiBox

The bounding box regression technique of SSD is inspired by Szegedys work on MultiBox, a method for fast class-agnostic bounding box coordinate proposals. Interestingly, in the work done on MultiBox an Inception-style convolutional network is used. The 1x1 convolutions that you see below help in dimensionality reduction since the number of dimensions will go down. MultiBoxs loss function also combined two critical components that made their way into SSD:

Confidence Loss: this measures how confident the network is of the objectness of the computed bounding box. Categorical cross-entropy is used to compute this loss.

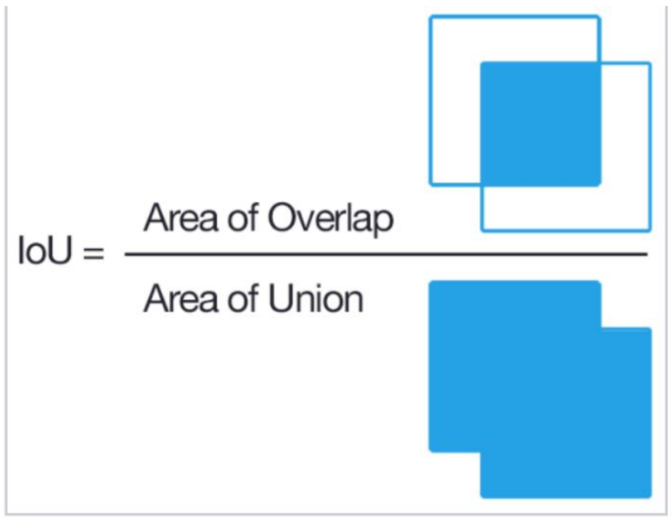
Location Loss: this measures how far away the networks predicted bounding boxes are from the ground truth ones from the training set. L2-Norm is used here.Without delving too deep into the math (read the paper if you are curious and want a more rigorous notation), the expression for the loss, which measures how far off our prediction landed, is thus.

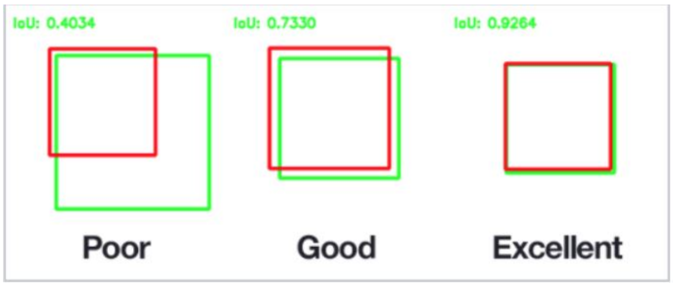
multiboxloss = confidenceloss + alpha * locationloss

The alpha term helps us in balancing the contribution of the location loss. As usual in deep learning, the goal is to find the parameter values that most optimally reduce the loss function, thereby bringing our predictions closer to the ground truth.

MultiBox Priors And IoU The logic revolving around the bounding box generation is actually more complex than what I earlier stated. But fear not: it is still within reach.

In MultiBox, the researchers created what we call priors (or anchors in Faster-R-CNN terminology), which are pre-computed, fixed size bounding boxes that closely match the distribution of the original ground truth boxes. In fact those priors are selected in such a way that their Intersection over Union ratio (aka IoU, and sometimes referred to as Jaccard index) is greater than 0.5. As you can infer from the image below, an IoU of 0.5 is still not good enough but it does however provide a strong starting point for the bounding box regression algorithmŁŁit is a much better strategy than starting the predictions with random coordinates! Therefore MultiBox starts with the priors as predictions and attempt to regress closer to the ground truth bounding boxes.





At the end, MultiBox only retains the top K predictions that have minimised both location (LOC) and confidence (CONF) losses.

C. Detection and Tracking

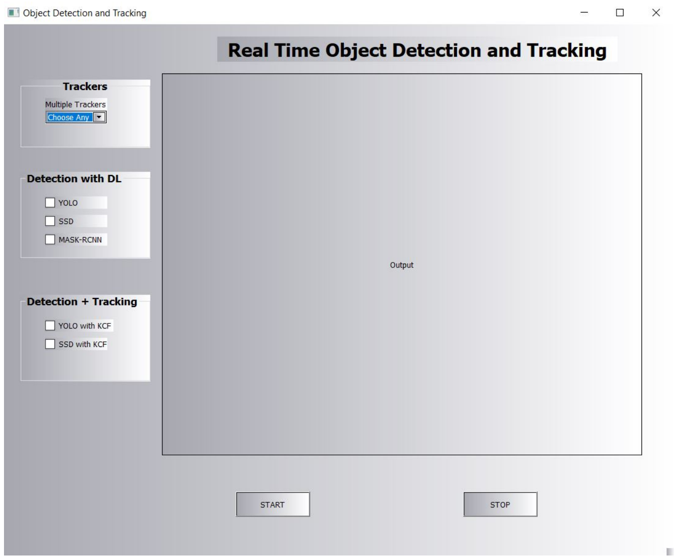
As of for now we have got good results for object detection using Deep learning Modules namely YOLO and SSD, and for tracking we used multiple trackers available in OpenCV library. We chose KCF tracker as it performs good and combined with detection outputs and performed Detection and Tracking at a same time.

III. Results

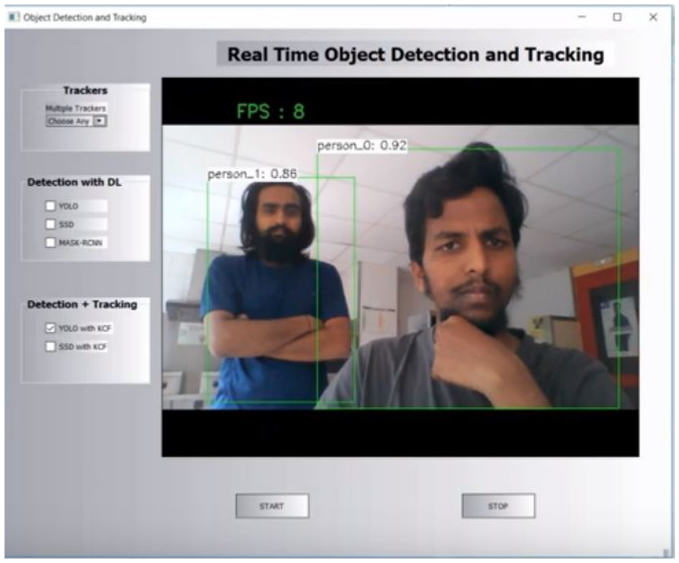
Follow the youtube link below for Video demonstration on how to run the code and to see the results.

`https://youtu.be/qULLaP_8X_Q`

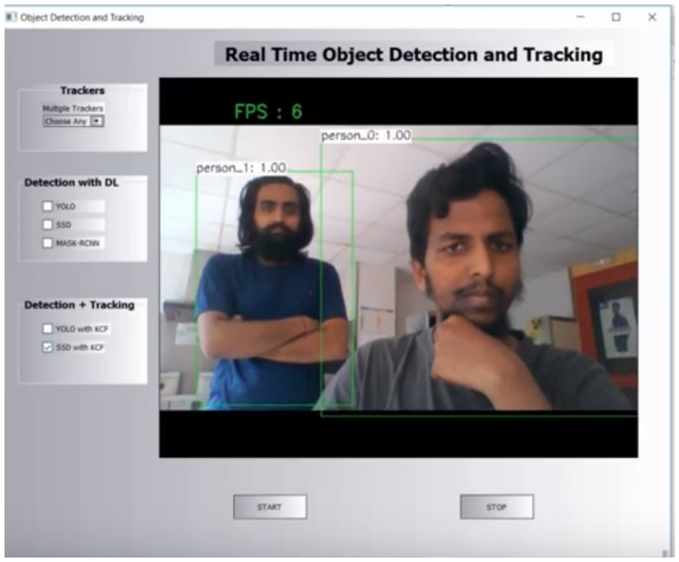
The GUI Design is as follows:



YOLO with KCF



SSD with KCF



References

[1] learnopencv.com.
[2] darknet.
[3] tensorflow library.
[4] OpenCV library.
[5] Medium.com
[6] towardsdatascience.com.
[7] hackernoon.com