Yolo for detecting cyrcles inside an eye

Massimiliano Finocchiaro massi.fino.001@gmail.com
Dragos Octavian Russo[1] russo.t.o.d@gmail.com

1. Introduction

During corneal surgery, it is challenging for the surgeon to accurately mend the patient's cornea. The surgeon must maintain a spherical shape of a small circle/ellipse projected through an LED onto the iris, while also keeping the eye hydrated to prevent drying, which can deform this circle.
Our approach aims to assist surgeons during corneal surgery. The difficulty lies in ensuring the seam of the iris is correctly aligned, and it can be hard to visually assess if the work is progressing correctly. We propose using a deep learning model based on YOLO to detect the circle within the iris and provide its dimensions in real-time. This allows the surgeon to easily monitor their work and ensure the cornea is mended accurately. By simplifying the process, we reduce the risk of human error due to fatigue and inattention, which is crucial in precision-demanding tasks.
For this project, we utilized a GitHub repository that contains a YOLO implementation [1] with Darknet for object classification across multiple classes. We modified the implementation to perform object classification on a single class (our circles).

2. Model Description

Our model is based on YOLO architecture, much layers are convolutional characterizated by: number of filter, size of filter, stride, padding and activation (Peaky Relu), another layer is shortcut layer that have the same function of these used in ResNet for deeper neural networks to avoid problem of vanishing gradient, in our case we sum the output of the third previous layer. Route layer is used to concatenate feature maps from different part of network, Upsampling layer is used for increase the resolution of feature maps. The YOLO layer used for detection of object. YOLOLayer takes as input the output of a previous layer and the size of image.It defines anchor box used to predict bounding box, number of class, type of loss used during train (mse loss and bce loss), during forward pass the output is adapted for predictions of bounding box and in reference create a grid for predictions. DarkNet is used to achive more computational efficency in convolutional operation. Optimization algorithms used is Adam, but is possible to use also SGD. Learning rate scheme initially uses a burn-in period, where the learning rate grows linearly until it reaches the specified initial value. Subsequently, the learning rate can be adapted based on the steps specified in the lr_steps list.
The main loss function in the code is customized and combined, based on the specifics of the object detection problem using YOLO (You Only Look Once). In the compute loss function, losses are calculated for each YOLO layer based on the model predictions and

desired targets. At the begin we setup the data in the build_targets function. We pick the targets and the anchors and transforms our annotations in a YOLO friendly format, because our annotations was referred to the whole pictures, so we need to adjust them, and we normalize them with the size of our anchors. This type of modifies were very usefull when we calculate the index of Union. Speaking of IoU we have different losses. The lbox represents the box regression loss, calculated as 1 minus the specified Index of Union (IoU). The lobj represents the object classification loss, calculated using the BCEWithLogitsLoss function.
In particular for our train we calculated CIoU that adds a penalty based on the distance between the centers of the bounding boxes and the appearance (ratio) of the bounding boxes. We can also choose to calculate GIoU and DIoU where GIoU adds a penalty based on the minimum area that contains both bounding boxes and DIoU adds a penalty based on the distance between the centers of the bounding boxes.

3. Dataset

- Data Source: We create a auto labeled dataset. Starting from a total of 50 videos, where we extracted all the frame and first we have annotated iris from frames for segmentation of iris, of this frames how second step we made a python script that create an artifical cyrcle/ellipse that we overlaid on the extracted frame for train YOLO model, This ellipse have different sizes, with border of 5 px, put in different parts of iris and we add from 3 to 5 of this ellipsis in all frame to train the model with objective of recognize the real cyrcle in the frame. After annotating them we split dataset in three parts: train, validation and test.

- Data Size: We used a dataset that contain a total of 20K annotated images, 70% for train, 20% for validation and 10% for test. In each image annotated we have the artificial circles,so we defined only one class in the mode.

- Preprocessing:

  First of all: due to the variety of image sizes we have put a function to resize images that were already squared to a size of 256x256

  For the creation of Dataset we took the image and its relative annotation and applied this transformations on train dataset:

  Sharpen: Increases the sharpness of the image by a factor between 0.0 and 0.1.

  Affine: Applies affine transformations with rotation, translation, and scale.

  translate_percent: Translation up to 10% of the image in both directions.

  scale: Scale the image with a factor between 0.8 and 1.5.

  AddToBrightness: Changes the brightness of the image with a value range between -60 and 40.

  AddToHue: Changes the image hue with a range of values between -10 and 10.

  Fliplr: Flip the image horizontally with a 50% chance.

  Resize transformation: Resize the image to the specified size (output size).

After we applied normalization to the coordinates of the bounding boxes with respect to the new dimensions of the image.

collate_fn where: Filter out invalid images. Separate images from bounding boxes. Stack images into a 4D tensor. Adds the sample index to the bounding boxes to identify the corresponding image. Concatenate all bounding boxes into a single 2D tensor. It returns the processed batch of images and bounding boxes, ready to be used in the model.
In the end, Dataloader is composed with batch of size 16 and is applied the function collate_fn defined before.

In validation dataset the difference is in the transformation where we apply only resize of image and normalization of coordinates.

4. Training details and procedure

1. **Evaluation Metrics**

   - Metrics: To evaluate the performance of model we measure precision, f1 score, recall and average precision. With precision we calculate the percentual of correct detection, with recall we estimate the percentual of correct prediction on the total number of true predictions, with average prediction we calculate the mean precision on different level of recall and the f1 score measure the armonic mean of precision and recall that allow us to obtain a balance of this metrics.

2. **Training procedure**

   - Epochs: 300
   - Batch Size: 16
   - Validation Strategy: The validation phase is processed each 5 epochs of train to see more change in metrics. For each batch in dataloader the target bounding boxes are converted from x_min, y_min, width, height format to x_min, y_min, x_max, y_max format using xywh2xyxy function. The bounding boxes are then scaled according to the image size. The model makes predictions for the batch of images with model(imgs) function. Non-Maximum Suppression (NMS) is applied to the predictions using non_max_suppression function to filter out redundant bounding boxes based on confidence and IoU thresholds. The function get_batch_statistics is called to compute true positives, predicted scores for each sample in the batch. This function iterates over each image in the batch, matches predicted boxes with ground truth boxes based on IoU, and determines true positives. The function ap_per_class is called to calculate average precision (AP), precision, recall, and F1 score for the class.
   - Stopping Criteria: We just have a checkpoint for any epoch trained but we don't define any stop criteria

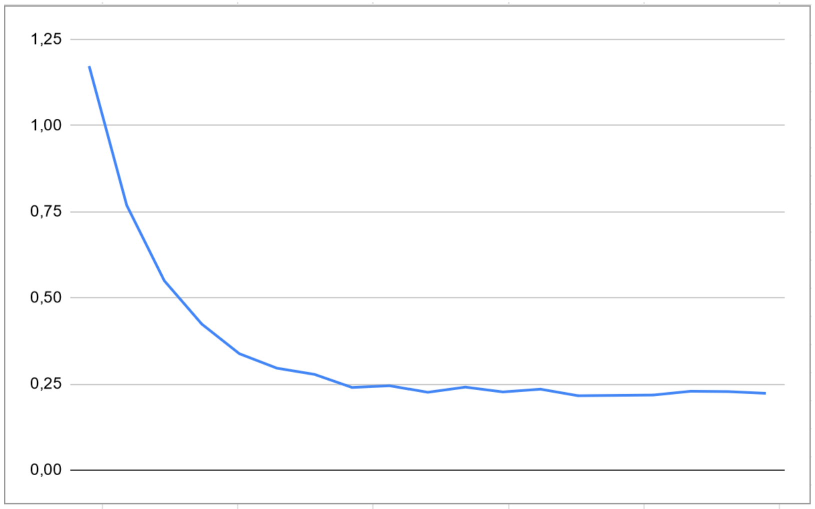
3. **Training details**

   - Optimizer: Optimization algorithms used are Adam.

- Hyperparameters: Learning rate scheme initially uses a burn-in period, where the learning rate grows linearly until it reaches the specific steps specified in the lr_steps list.d initial value. Subsequently, the learning rate can be adapted based on the steps specified in the lr_steps list. We iterate a batch of size 16 for a total of 906 batch in train and 231 in validation for 300 epochs.
- Initialization: For convolution (Conv) modules, the weights are initialized with a normal (Gaussian) distribution with mean 0 and standard deviation 0.02. For the batch normalization modules (BatchNorm2d), the weights are initialized with a normal distribution with mean 1 and standard deviation 0.02. The biases of the batch normalization modules are initialized to 0. The initialization of the weights with mean 1 is chosen because the batch norm scaling parameters start at 1 (no scaling at the beginning) and the bias at 0 (no shift at the start), which is a standard choice for batch normalization, where scale (γ, "gamma") and bias (β, "beta"). This allows batch normalization to start with a neutral effect on the data distribution, letting the network gradually learn the appropriate transformations during training.

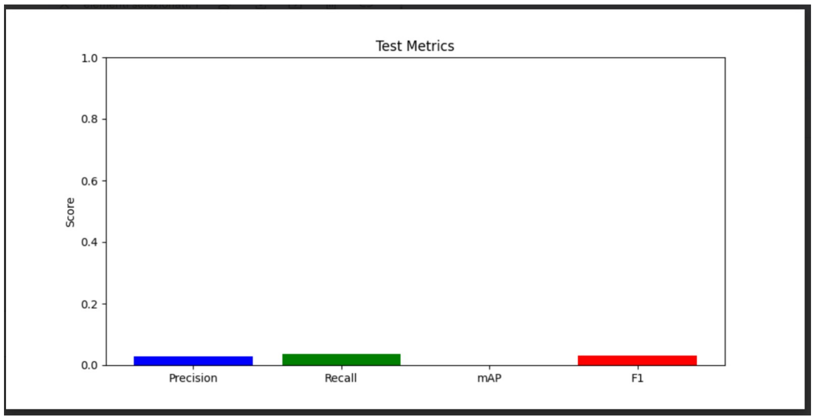
5. Experimental Results

- Training Curves:



On loss curve we have a good loss minimization starts from a low numbers of epochs. Continuing training we see that the loss will go down and down again but not with the same efficency we obtained at the starting phase of the train. As the lowest values we reached nearly 0,20 after 250+ epochs of train.

- Performance: We calculate test performance with score of f1, Recall, precision and average precision, but due to lackness of computing power we don't get good performance, but we saw a bit of improvment in performance on confidence and we espect that after much time of train we can arrive to high metrics value.



This is an example of our metrics values that we obtain; the scores are very low.

References

[1] inference A minimal PyTorch implementation of YOLOv3 with support for training and evaluation. “Deep Residual Learning for Image Recognition”. In: (2021). URL: https://github.com/eriklindernoren/PyTorch-YOLOv3.