Project #3 – Object Detection using YOLOv3

Mark Klaiman, 208734715

+

Omri Gruman, 318965621

=========

527700336

1. Introduction

In the final project we attempt to use a pre-trained object detection model on a different dataset in order to apply transfer learning to a different task. The architecture that we chose is YOLOv3 architecture, on which we will elaborate soon. After choosing the model, we set up our training environment, start the training process, followed by a brief evaluation of some test predictions, and end up by detecting objects in a video that displayed a variety of poker cards.

2. YOLOv3

YOLOv3 is a deep learning model for detecting the position and the type of an object from the input image. It can classify objects in one of the 80 categories available (e.g., car, person, motorbike…), and compute bounding boxes for those objects from a single input image.

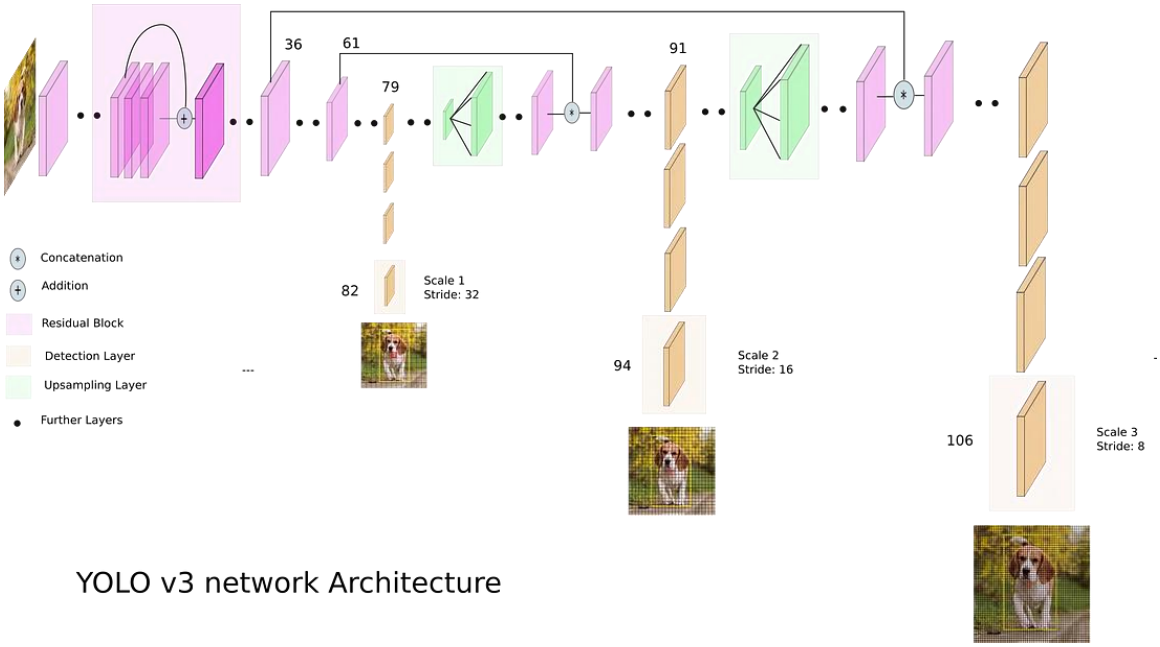
2.1. Architecture

The general YOLO architecture introduced a new approach to object detection, in which the feature extraction and object localization were unified into a single monolithic block. Furthermore — the localization and classification heads were also united. This single-stage architecture, named ***YOLO*** (You Only Look Once) results in a very fast inference time.

The algorithm separates an image into a grid. Each grid cell predicts up to 3 boundary boxes (also referred to as anchor boxes) around objects that score highly with the aforementioned predefined classes. Each boundary box has a respective confidence score of how accurate it assumes that prediction should be and detects only one object per bounding box.

Inspired by ***ResNet*** and ***FPN*** (Feature-Pyramid Network) architectures, *YOLO-V3 feature extractor*, called ***Darknet-53***, contains skip connections (like ResNet) and 3 prediction heads (like FPN) — each processing the image at a different spatial compression.

A Feature-Pyramid is a topology developed in 2017 by Facebook A.I. Research in which the feature map gradually decreases in spatial dimension, but later the feature map expands again and is concatenated with previous feature maps with corresponding sizes. This procedure is repeated, and each concatenated feature map is fed to a separate detection head, allowing the YOLOv3 to learn objects at different sizes.

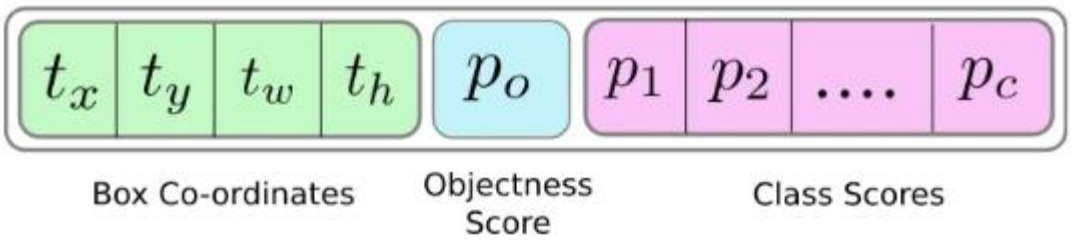


YOLO v3 network Architecture

2.2. Outputs

Each cell in the output layer's feature map predicts 3 boxes in the case of YOLOv3 — one box per anchor. Each box prediction consists of:

- (x, y) for box center (between 0 and 1, relative to cell center),
- (w, h) for box size scales (relative to anchor dimensions),
- confidence value for objectness score (between 0 and 1),
- 'number-of-classes' values for class score (each between 0 and 1)



2.3. Loss function

The YOLO loss for each box prediction is comprised of the following terms:

- **Coordinate loss** – Mean Squared Error of the X coordinate, Y coordinate, width, and height of each bounding box.
- **Objectness loss** – Binary Cross-Entropy of the confidence score of each bounding box.
- **Classification loss** – Binary Cross-Entropy of the class prediction scores of each bounding box.

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

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

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

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

For each spatial cell, for each box prediction in centered in that cell, the loss function finds the box with the best IoU with the object centered in that cell. This distinguishing mechanism between ***best boxes*** and all the other boxes is in the heart of the YOLO loss.

The objectness loss term teaches the network to predict a correct IoU, while the coordinate loss teaches the network to predict a better box (which eventually pushes the IoU toward 1.0). All the box predictions contribute to the objectness loss, but only the best-fitting boxes in each spatial cell also contribute a coordinate and classification loss.

2.4. Inference

At inference we usually have multiple boxes with varying coverage for each object. We want the post-processing algorithm to choose the box that covers the object in the most exact way. We also want to choose the box that gives the correct class prediction for the object.

Firstly, the objectness tells us how good the coverage is — so boxes with very small objectness are discarded and don't even make it to the NMS block. Secondly, NMS is done for each class separately, so the class score is scaled by the box objectness for a meaningful comparison.

3. Fine-tuning

After exploring the YOLOv3 model, we built a training pipeline to fine-tune its pre-trained weights to detect different objects from our custom dataset.

3.1. Setup

Throughout the training process we used the Ultralytics implementation to the YOLOv3 model, from their GitHub repository. So, the first thing we did is clone their repository.

3.2. Our data

We explored many different datasets in the public Roboflow object detection dataset archive. Finally, we chose a poker cards datasets which contains 52 different classes, 1 for each unique card. Our dataset includes almost 1300 images of poker cards in different angles, lightings

and positions, and each card appeared between 100 and 120 times in the dataset.

Then, we rearranged the data directories to match the tree that was expected by the implemented YOLOv3 code, and created a configuration file to let the training module know where the data images reside.

3.3. Training

During the training phase, we simply ran the training module on our dataset, using the YOLOv3 architecture and its pre-trained weights on the COCO dataset. We tried several times to freeze some layers of the model, but it turned out that the best performance was achieved without freezing any of the layers at all.

After the training finished, we visualized some results that included the different loss components, the Recall & Precision metrics as well as the mAP metric on a single IoU and on a range of IoU values. At last, we loaded the TensorBoard module to explore the results even further.

3.4. Evaluation

To sum up the pipeline, we used the detection module to try and detect the different poker card classes in the test set, and plotted them to try and see how successful was our training.

The final product was a video that showcased a variety of poker cards in which we tried to detect the different types of cards and plot the bounding boxes around each of the detected card.

4. Conclusion

In conclusion, we applied fine-tuning to the YOLOv3 model using a poker card dataset. Later, we constructed a training pipeline to easily go through all the stages of the training process, and finally we used the fine-tuned model to detect some poker card in a cool video.