DELHI TECHNOLOGICAL UNIVERSITY

Machine Learning Project Report

Video Content Analysis (Object Detection)

Under supervision of: Dr. Anil Singh Parihar

Submitted by:

PRAJJWAL CHITTORI (2K18/CO/249)

(Dept. of Computer Science and Engineering)

Introduction

Video content analysis being defined as the ability of analysing and deconstructing various temporal and spatial events in a video. Deep Learning techniques are well efficient in image analysis but video analysis is a rather time-consuming chore for a machine considering the huge amount of data to be analysed by the system. Content analysis would pave the way for further operations such as video classification, video structuring just to name a few. As the world advancing towards more streaming based networking contributed by the rise of video streaming platform giants such as Netflix and Amazon prime contributes to the rise of various issues such as video content classification, identification and behaviour analysis etc.

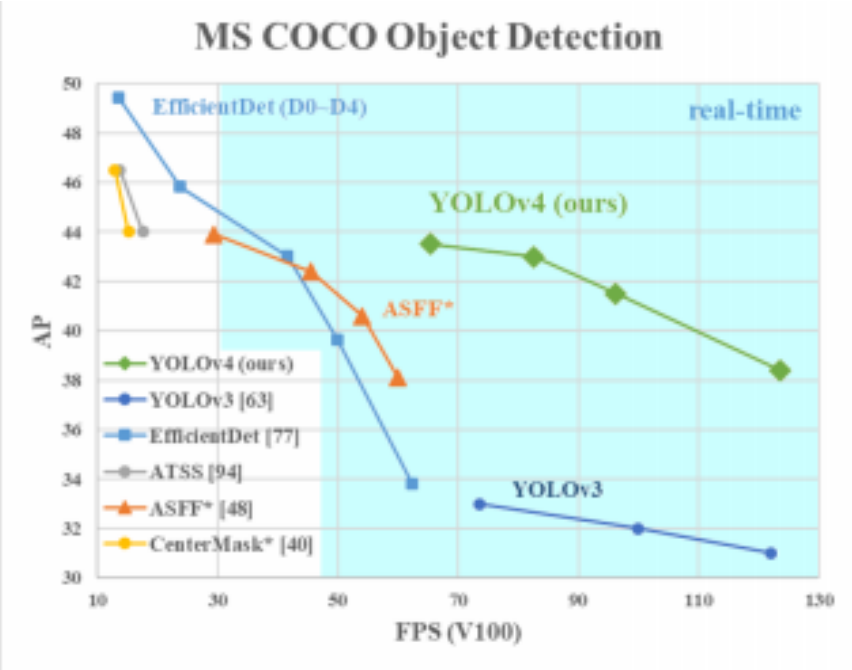
Convolution Neural Networks comes under deep neural networks that are used to analyse visual information and imagery. They are based on shared-weights architecture and translation invariance charactertics. Yolo(you only look once) is convolution neural network model used for object identification in images and videos. The approach for Improving the algorithm includes testing of features on large datasets, with features such as batch-normalization and residual-connection.

We'll work on features and connections such as Weighted Residual Connections (WRC), Cross Stage Partial connections (CSP), Cross mini Batch Normalization (CmBN), Self-adversarial-training (SAT) and Mish-activation.

Accurate models for object identification are efficient but they are slow that is why they are used in instances where slow identification is needed such as video surveillance system analysis etc. Situations where fast object identification Is needed such as car collision warning systems the models are inaccurate because of lack of efficient algorithms and lack of computational power for such minute time instances. Modern accurate object identification models work not in real time and require large computational and GPU processing.

YOLO algorithm creates a convolutional neural network that works on only one GPU that works on a Modern Personal System.

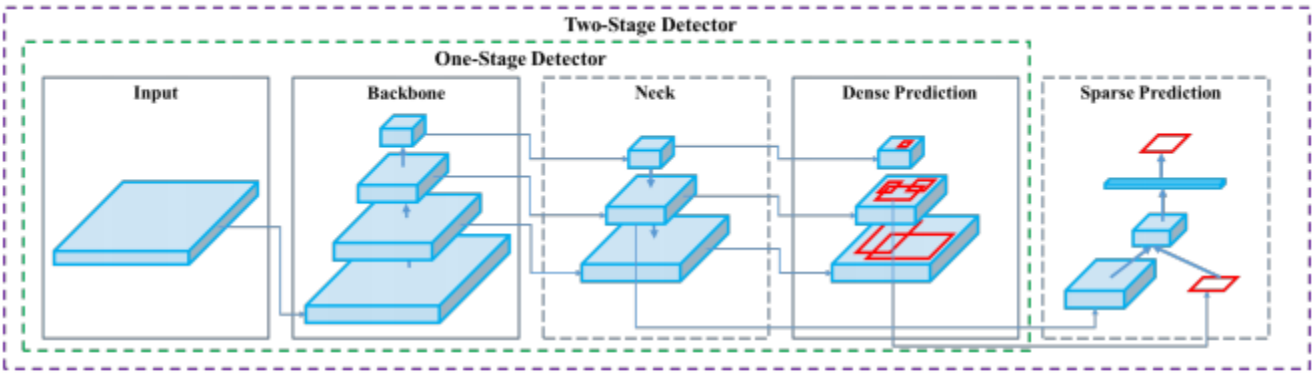
Here is the performance comparison of YOLO with other modern Object Detection models.



The main goal is designing a fast object detection model and optimize parallel computing systems.

The main features that are included in this version over the previous YOLO v3 are-

- The model should be able to trainable on modern conventional personal computing GPUs such as Nvidia 2080 Ti and other AMD equivalents.
- Working on Bag of specials method for object detection.
- Implementation of CBN,PAN and SAM that are easier to execute on slower systems.



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Literature review

The research paper that were referenced and studied to implement the YOLO algorithm are-

- **YOLOv4: Optimal Speed and Accuracy of Object Detection(**Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark)
- **You Only Look Once: Unified, Real-Time Object Detection**(Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi)
- **Object Detection with Deep Learning: A Review**(Zhong-Qiu Zhao, Shou-tao Xu, Xindong Wu)

In the following papers they work on deep learning-based object detection system.

It has been proposed to use Convolutional neural network system. These papers have worked on this basic CNN with many modifications and feature implementations to make this network work faster.

The system works on unifying several components of object detection

into a single convolutional neural network. The network implements features

from every single frame of the video or several images to construct each bounding

box by prediction. This will also be implemented by predicting bounding boxes across all classes for a video. This results in the network reasoning globally about the full image and all the objects in the image.

The YOLO also works on end-to-end training and on time analysis while maintaining high and accurate precision.

The system will take a video frame by frame and convert each frame into SxS grid.

If the centre of the object to be detected falls into the centre to be detected then it is responsible for detection.

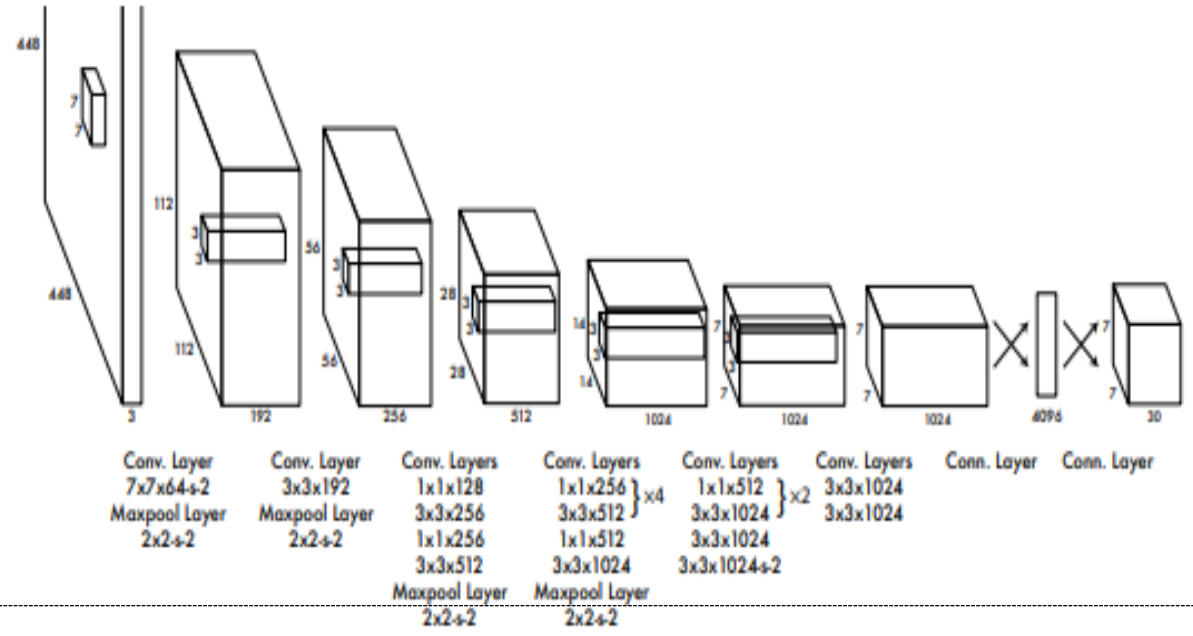
Each grid will then be responsible for predicting the bounding boxes and as result predicts the confidence scores for the system.

This paper also discusses further about the general performance of YOLO. It performs better than other modern algorithms such as DPM and RCNN, by a long shot when generalising on standard people art dataset.

The present paper discusses upon the implementation and various fixes to improve the performance of YOLO algorithm.

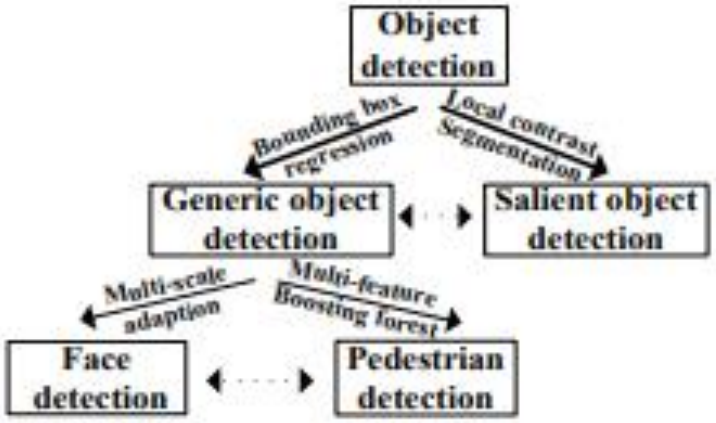
Theory

“You Only Look Once” implements a model that uses convolutional neural networks for video analysis in the field of object detection. You only look once, or



YOLO outperforms many major modern object detection algorithms. It works on the principle of fast object detection but sometimes there are instances of not that accurate results provided by the model because of the speed constraint, but generally it is accurate enough for real time fast object detection systems.

Contrasting with recognition algorithms, a YOLO algorithm does not only predict labels of class but locations of objects as well. So, it can predict multiple objects in an image or frame by frame broken video file. YOLO Algorithm implements a single Convolutional Neural network to the Full Video. The network divides the video into images and similarly image into regions and constructs the bounding boxes and probabilities for each region. These bounding boxes are then implemented by being weighted by the predicted probabilities.



YOLO comprises of:

- Backbone: **CSPDarknet**
- Neck: **SPP , PAN**
- Head: **YOLOv3**

Uses:

- **Bag of Freebies method (BoF) for backbone:**

CutMix and Mosaic implementation and data augmentation, DropBlock analysis

and regularization, label smoothing

- **Bag of Specials method(BoS) for backbone:**

Mish calculation and activation, Methods used such as Cross-stage partial connections method (CSP),and Multi-input weighted included with residual connections (MiWRC)

- **Bag of Freebies method(BoF) for detector:**

CIoU-loss, CmBN, DropBlock analysis and regularization, Mosaic implementation and data augmentation, Adversarial Training, grid sensitivity elimination, multiple angle creation for single frames, Cosine annealing scheduler , Optimal and accurate hyperparameters, Random training and model construction

- **Bag of Specials method (BoS) for detector:**

Mish calculation and activation, SPP-block implementation , SAM-block implementation , PAN path and aggregation block, DIoU-NMS

YOLO works by predicting multiple bounding boxes over grid cell. During training time one only wants that one bounding box predictor to be referenced per each object. The algorithm assigns one predictor to be in the work of predicting an object due to which the result has the largest current IOU with the real object. This leads to analysing differences and specialization between bounding box predictors. After successive turns each predictor gets better at analyzing certain shapes, aspect and metric ratios, or classes of object, improving overall quality of prediction.

The loss function implemented is-

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

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

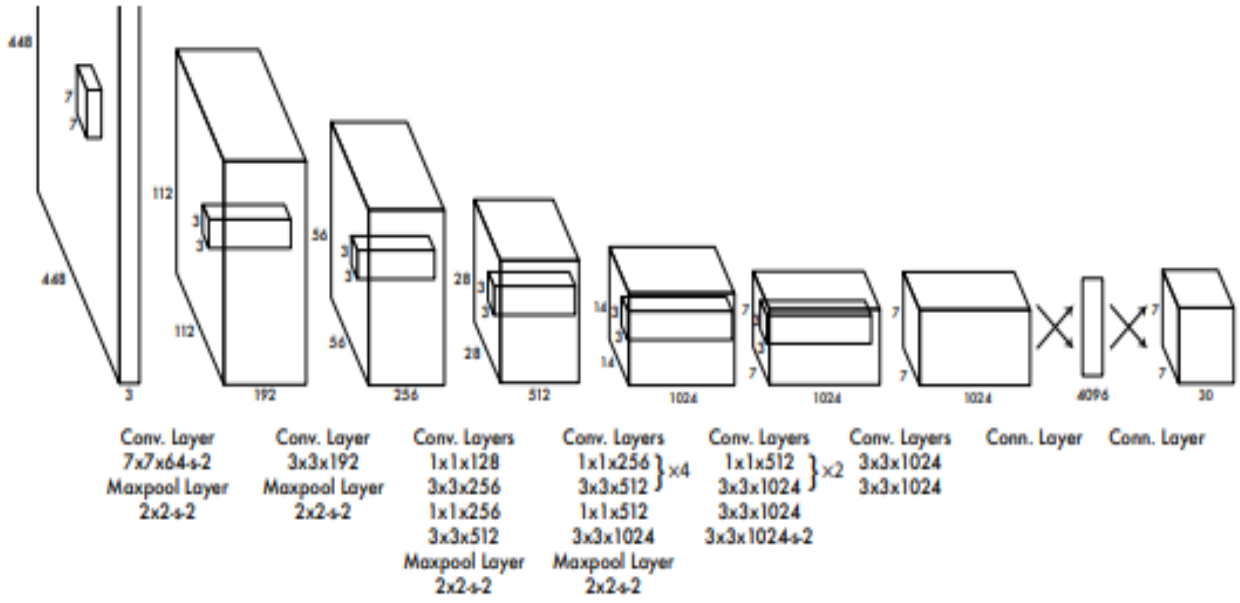
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

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

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

This paper implements a convolutional neural network and test it on the PASCAL VOC detection dataset. The initial convolutional layers hacks the features from the image while the connected layers of the bounding box predict the probabilistic and coordinate data. The network architecture is based on GoogLeNet model for video analysis. The network has 24 layers of convolutional network and 2 fully connected layers . Not using the inception modules of GoogLeNet, This implementation will use simply use 1x1 reduction layers with 3x3 convolutional layers. This implementation will also train a fast version of YOLO used for analysis of fast moving objects in videos. Fast implementation of YOLO uses a neural network with low convolutional layers (9) and lower filters in those layers. Contrasting with the size of the network , training and testing modules, data sets and parameters are the similar in YOLO and Fast YOLO.

The image describing the model is-



The code describing the model is as follows-

The pre initialization for the model

```
from ctypes import *
import math
import random
import os


class BOX(Structure):
    _fields_ = [("x", c_float),
                ("y", c_float),
                ("w", c_float),
                ("h", c_float)]


class DETECTION(Structure):
    _fields_ = [("bbox", BOX),
                ("classes", c_int),
                ("prob", POINTER(c_float)),
                ("mask", POINTER(c_float)),
                ("objectness", c_float),
                ("sort_class", c_int),
                ("uc", POINTER(c_float)),
                ("points", c_int),
                ("embeddings", POINTER(c_float)),
                ("embedding_size", c_int),
                ("sim", c_float),
                ("track_id", c_int)]
```

```
class DETNUMPAIR(Structure):
    _fields_ = [("num", c_int),
                ("dets", POINTER(DETECTION))]


class IMAGE(Structure):
    _fields_ = [("w", c_int),
                ("h", c_int),
                ("c", c_int),
                ("data", POINTER(c_float))]


class METADATA(Structure):
    _fields_ = [("classes", c_int),
                ("names", POINTER(c_char_p))]


def network_width(net):
    return lib.network_width(net)


def network_height(net):
    return lib.network_height(net)
```

Network Creation

```
def load_network(config_file, data_file, weights, batch_size=1):
    """
    load model description and weights from config files
    args:
        config_file (str): path to .cfg model file
        data_file (str): path to .data model file
        weights (str): path to weights
    returns:
        network: trained model
        class_names
        class_colors
    """
    network = load_net_custom(
        config_file.encode("ascii"),
        weights.encode("ascii"), 0, batch_size)
    metadata = load_meta(data_file.encode("ascii"))
    class_names = [metadata.names[i].decode("ascii") for i in range(metadata.classes)]
    colors = class_colors(class_names)
    return network, class_names, colors


def print_detections(detections, coordinates=False):
    print("\nObjects:")
    for label, confidence, bbox in detections:
        x, y, w, h = bbox
        if coordinates:
            print("{}: {}%    (left_x: {:.0f}   top_y:  {:.0f}   width:   {:.0f}   height:  {:.0f})".format(label, confidence, x, y, w, h))
        else:
            print("{}: {}%".format(label, confidence))


def draw_boxes(detections, image, colors):
    import cv2
    for label, confidence, bbox in detections:
        left, top, right, bottom = bbox2points(bbox)
```

Bouding Box Generation

```
def draw_boxes(detections, image, colors):
    import cv2
    for label, confidence, bbox in detections:
        left, top, right, bottom = bbox2points(bbox)
        cv2.rectangle(image, (left, top), (right, bottom), colors[label], 1)
        cv2.putText(image, "{} [{:.2f}]".format(label, float(confidence)),
                    (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    colors[label], 2)
    return image


def decode_detection(detections):
    decoded = []
    for label, confidence, bbox in detections:
        confidence = str(round(confidence * 100, 2))
        decoded.append((str(label), confidence, bbox))
    return decoded


def remove_negatives(detections, class_names, num):
    """
    Remove all classes with 0% confidence within the detection
    """
    predictions = []
    for j in range(num):
        for idx, name in enumerate(class_names):
            if detections[j].prob[idx] > 0:
                bbox = detections[j].bbox
                bbox = (bbox.x, bbox.y, bbox.w, bbox.h)
                predictions.append((name, detections[j].prob[idx], (bbox)))
    return predictions
```

Object Detection

```python
def decode_detection(detections):
    decoded = []
    for label, confidence, bbox in detections:
        confidence = str(round(confidence * 100, 2))
        decoded.append((str(label), confidence, bbox))
    return decoded


def remove_negatives(detections, class_names, num):
    """
    Remove all classes with 0% confidence within the detection
    """
    predictions = []
    for j in range(num):
        for idx, name in enumerate(class_names):
            if detections[j].prob[idx] > 0:
                bbox = detections[j].bbox
                bbox = (bbox.x, bbox.y, bbox.w, bbox.h)
                predictions.append((name, detections[j].prob[idx], (bbox)))
    return predictions


def detect_image(network, class_names, image, thresh=.5, hier_thresh=.5, nms=.45):
    """
        Returns a list with highest confidence class and their bbox
    """
    pnum = pointer(c_int(0))
    predict_image(network, image)
    detections = get_network_boxes(network, image.w, image.h,
                                   thresh, hier_thresh, None, 0, pnum, 0)
    num = pnum[0]
    if nms:
        do_nms_sort(detections, num, len(class_names), nms)
    predictions = remove_negatives(detections, class_names, num)
    predictions = decode_detection(predictions)
```

```
hasGPU = True
if os.name == "nt":
    cwd = os.path.dirname(__file__)
    os.environ['PATH'] = cwd + ';' + os.environ['PATH']
    winGPUdll = os.path.join(cwd, "yolo_cpp_dll.dll")
    winNoGPUdll = os.path.join(cwd, "yolo_cpp_dll_nogpu.dll")
    envKeys = list()
    for k, v in os.environ.items():
        envKeys.append(k)
    try:
        try:
            tmp = os.environ["FORCE_CPU"].lower()
            if tmp in ["1", "true", "yes", "on"]:
                raise ValueError("ForceCPU")
            else:
                print("Flag value {} not forcing CPU mode".format(tmp))
        except KeyError:
            # We never set the flag
            if 'CUDA_VISIBLE_DEVICES' in envKeys:
                if int(os.environ['CUDA_VISIBLE_DEVICES']) < 0:
                    raise ValueError("ForceCPU")
            try:
                global DARKNET_FORCE_CPU
                if DARKNET_FORCE_CPU:
                    raise ValueError("ForceCPU")
            except NameError as cpu_error:
                print(cpu_error)
        if not os.path.exists(winGPUdll):
            raise ValueError("NoDLL")
        lib = CDLL(winGPUdll, RTLD_GLOBAL)
    except (KeyError, ValueError):
        hasGPU = False
        if os.path.exists(winNoGPUdll):
```

```
else:
    lib = CDLL(os.path.join(
        os.environ.get('DARKNET_PATH', './'),
        "libdarknet.so"), RTLD_GLOBAL)
lib.network_width.argtypes = [c_void_p]
lib.network_width.restype = c_int
lib.network_height.argtypes = [c_void_p]
lib.network_height.restype = c_int

copy_image_from_bytes = lib.copy_image_from_bytes
copy_image_from_bytes.argtypes = [IMAGE,c_char_p]

predict = lib.network_predict_ptr
predict.argtypes = [c_void_p, POINTER(c_float)]
predict.restype = POINTER(c_float)

if hasGPU:
    set_gpu = lib.cuda_set_device
    set_gpu.argtypes = [c_int]

init_cpu = lib.init_cpu

make_image = lib.make_image
make_image.argtypes = [c_int, c_int, c_int]
make_image.restype = IMAGE

get_network_boxes = lib.get_network_boxes
get_network_boxes.argtypes = [c_void_p, c_int, c_int, c_float, c_float, POINTER(c_int), c_int, POINTER(c_int), c_int]
get_network_boxes.restype = POINTER(DETECTION)

make_network_boxes = lib.make_network_boxes
make_network_boxes.argtypes = [c_void_p]
make_network_boxes.restype = POINTER(DETECTION)

free_detections = lib.free_detections
free_detections.argtypes = [POINTER(DETECTION), c_int]
```

```
free_ptrs = lib.free_ptrs
free_ptrs.argtypes = [POINTER(c_void_p), c_int]

network_predict = lib.network_predict_ptr
network_predict.argtypes = [c_void_p, POINTER(c_float)]

reset_rnn = lib.reset_rnn
reset_rnn.argtypes = [c_void_p]

load_net = lib.load_network
load_net.argtypes = [c_char_p, c_char_p, c_int]
load_net.restype = c_void_p

load_net_custom = lib.load_network_custom
load_net_custom.argtypes = [c_char_p, c_char_p, c_int, c_int]
load_net_custom.restype = c_void_p

free_network_ptr = lib.free_network_ptr
free_network_ptr.argtypes = [c_void_p]
free_network_ptr.restype = c_void_p

do_nms_obj = lib.do_nms_obj
do_nms_obj.argtypes = [POINTER(DETECTION), c_int, c_int, c_float]

do_nms_sort = lib.do_nms_sort
do_nms_sort.argtypes = [POINTER(DETECTION), c_int, c_int, c_float]

free_image = lib.free_image
free_image.argtypes = [IMAGE]

letterbox_image = lib.letterbox_image
letterbox_image.argtypes = [IMAGE, c_int, c_int]
letterbox_image.restype = IMAGE

load_meta = lib.get_metadata
```

```
do_nms_sort = lib.do_nms_sort
do_nms_sort.argtypes = [POINTER(DETECTION), c_int, c_int, c_float]

free_image = lib.free_image
free_image.argtypes = [IMAGE]

letterbox_image = lib.letterbox_image
letterbox_image.argtypes = [IMAGE, c_int, c_int]
letterbox_image.restype = IMAGE

load_meta = lib.get_metadata
lib.get_metadata.argtypes = [c_char_p]
lib.get_metadata.restype = METADATA

load_image = lib.load_image_color
load_image.argtypes = [c_char_p, c_int, c_int]
load_image.restype = IMAGE

rgbgr_image = lib.rgbgr_image
rgbgr_image.argtypes = [IMAGE]

predict_image = lib.network_predict_image
predict_image.argtypes = [c_void_p, IMAGE]
predict_image.restype = POINTER(c_float)

predict_image_letterbox = lib.network_predict_image_letterbox
predict_image_letterbox.argtypes = [c_void_p, IMAGE]
predict_image_letterbox.restype = POINTER(c_float)

network_predict_batch = lib.network_predict_batch
network_predict_batch.argtypes = [c_void_p, IMAGE, c_int, c_int, c_int,
                                  c_float, c_float, POINTER(c_int), c_int, c_int]
network_predict_batch.restype = POINTER(DETNUMPAIR)
```

The code used to design and implement video analysing capabilities to the model are as follows-

Initialization

```
from ctypes import *
import random
import os
import cv2
import time
import darknet
import argparse
from threading import Thread, enumerate
from queue import Queue


def parser():
    parser = argparse.ArgumentParser(description="YOLO Object Detection")
    parser.add_argument("--input", type=str, default=0,
                        help="video source. If empty, uses webcam 0 stream")
    parser.add_argument("--out_filename", type=str, default="",
                        help="inference video name. Not saved if empty")
    parser.add_argument("--weights", default="yolov4.weights",
                        help="yolo weights path")
    parser.add_argument("--dont_show", action='store_true',
                        help="windown inference display. For headless systems")
    parser.add_argument("--ext_output", action='store_true',
                        help="display bbox coordinates of detected objects")
    parser.add_argument("--config_file", default="./cfg/yolov4.cfg",
                        help="path to config file")
    parser.add_argument("--data_file", default="./cfg/coco.data",
                        help="path to data file")
    parser.add_argument("--thresh", type=float, default=.25,
                        help="remove detections with confidence below this value")
    return parser.parse_args()


def str2int(video_path):
    """
```

Inference and analysis functions

```
        except ValueError:
            return video_path


def check_arguments_errors(args):
    assert 0 < args.thresh < 1, "Threshold should be a float between zero and one (non-inclusive)"
    if not os.path.exists(args.config_file):
        raise(ValueError("Invalid config path {}".format(os.path.abspath(args.config_file))))
    if not os.path.exists(args.weights):
        raise(ValueError("Invalid weight path {}".format(os.path.abspath(args.weights))))
    if not os.path.exists(args.data_file):
        raise(ValueError("Invalid data file path {}".format(os.path.abspath(args.data_file))))
    if str2int(args.input) == str and not os.path.exists(args.input):
        raise(ValueError("Invalid video path {}".format(os.path.abspath(args.input))))


def set_saved_video(input_video, output_video, size):
    fourcc = cv2.VideoWriter_fourcc(*"MJPG")
    fps = int(input_video.get(cv2.CAP_PROP_FPS))
    video = cv2.VideoWriter(output_video, fourcc, fps, size)
    return video


def video_capture(frame_queue, darknet_image_queue):
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame_resized = cv2.resize(frame_rgb, (width, height),
                                   interpolation=cv2.INTER_LINEAR)
        frame_queue.put(frame_resized)
        darknet.copy_image_from_bytes(darknet_image, frame_resized.tobytes())
        darknet_image_queue.put(darknet_image)
    cap.release()
```

```python
def inference(darknet_image_queue, detections_queue, fps_queue):
    while cap.isOpened():
        darknet_image = darknet_image_queue.get()
        prev_time = time.time()
        detections = darknet.detect_image(network, class_names, darknet_image, thresh=args.thresh)
        detections_queue.put(detections)
        fps = int(1/(time.time() - prev_time))
        fps_queue.put(fps)
        print("FPS: {}".format(fps))
        darknet.print_detections(detections, args.ext_output)
        darknet.free_image(darknet_image)
    cap.release()


def drawing(frame_queue, detections_queue, fps_queue):
    random.seed(3)  # deterministic bbox colors
    video = set_saved_video(cap, args.out_filename, (width, height))
    while cap.isOpened():
        frame_resized = frame_queue.get()
        detections = detections_queue.get()
        fps = fps_queue.get()
        if frame_resized is not None:
            image = darknet.draw_boxes(detections, frame_resized, class_colors)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            if args.out_filename is not None:
                video.write(image)
            if not args.dont_show:
                cv2.imshow('Inference', image)
            if cv2.waitKey(fps) == 27:
                break
    cap.release()
    video.release()
    cv2.destroyAllWindows()
```

Main Function

```python
if __name__ == '__main__':
    frame_queue = Queue()
    darknet_image_queue = Queue(maxsize=1)
    detections_queue = Queue(maxsize=1)
    fps_queue = Queue(maxsize=1)

    args = parser()
    check_arguments_errors(args)
    network, class_names, class_colors = darknet.load_network(
            args.config_file,
            args.data_file,
            args.weights,
            batch_size=1
        )
    # Darknet doesn't accept numpy images.
    # Create one with image we reuse for each detect
    width = darknet.network_width(network)
    height = darknet.network_height(network)
    darknet_image = darknet.make_image(width, height, 3)
    input_path = str2int(args.input)
    cap = cv2.VideoCapture(input_path)
    Thread(target=video_capture, args=(frame_queue, darknet_image_queue)).start()
    Thread(target=inference, args=(darknet_image_queue, detections_queue, fps_queue)).start()
    Thread(target=drawing, args=(frame_queue, detections_queue, fps_queue)).start()
```

Comparative Analysis

Progressive element portrayal, which is the staggered portrayals from pixel to elevated level semantic highlights learned by a progressive multi-stage structure, can be gained from information consequently and concealed variables of information can be unravelled through staggered nonlinear mappings.

• Compared with conventional shallow models, a more profound design gives a dramatically expanded expressive ability.

• The engineering of CNN gives an occasion to mutually upgrade a few related undertakings together (for example Quick RCNN consolidates classification and bouncing box relapse into a perform various tasks inclining way).

• Benefitting from the enormous learning limit of profound CNNs, some old-style PC vision difficulties can be reworked as high-dimensional information change issues and settled from an alternate perspective.

The mix of article locators and superpixel classification structure increases a promising outcome on semantic scene division task. Ouyang et al. proposed a deformable profound CNN (DeepID-Net) which presents a novel disfigurement compelled pooling (def-pooling) layer to force mathematical punishment on the distortion of different item parts and makes an outfit of models with various settings. Lenc et al. gave an examination on the function of proposition age in CNN-based indicators and attempted to supplant this stage with a steady and trifling locale age conspire. The objective is accomplished by biasing examining to coordinate the measurements of the ground truth jumping boxes with K-implies grouping. Nonetheless, more competitor boxes are needed to accomplish equivalent outcomes to those of R-CNN.

Overview of modern object identification architectures

Framework	Proposal	Multi-scale Input	Learning Method	Loss Function	Softmax Layer	End-to-end Train	Platform	Language
R-CNN [15]	Selective Search	-	SGD,BP	Hinge loss (classification),Bounding box regression	+	-	Caffe	Matlab
SPP-net [64]	EdgeBoxes	+	SGD	Hinge loss (classification),Bounding box regression	+	-	Caffe	Matlab
Fast RCNN [16]	Selective Search	+	SGD	Class Log loss+bounding box regression	+	-	Caffe	Python
Faster R-CNN [18]	RPN	+	SGD	Class Log loss+bounding box regression	+	+	Caffe	Python/Matlab
R-FCN [65]	RPN	+	SGD	Class Log loss+bounding box regression	-	+	Caffe	Matlab
Mask R-CNN [67]	RPN	+	SGD	Class Log loss+bounding box regression +Semantic sigmoid loss	+	+	TensorFlow/Keras	Python
FPN [66]	RPN	+	Synchronized SGD	Class Log loss+bounding box regression	+	+	TensorFlow	Python
YOLO [17]	-	-	SGD	Class sum-squared error loss+bounding box regression +object confidence+background confidence	+	+	Darknet	C
SSD [71]	-	-	SGD	Class softmax loss+bounding box regression	-	+	Caffe	C++
YOLOv2 [72]	-	-	SGD	Class sum-squared error loss+bounding box regression +object confidence+background confidence	+	+	Darknet	C

* '+' denotes that corresponding techniques are employed while '-' denotes that this technique is not considered. It should be noticed that R-CNN and SPP-net can not be trained end-to-end with a multi-task loss while the other architectures are based on multi-task joint training. As most of these architectures are re-implemented on different platforms with various programming languages, we only list the information associated with the versions by the referenced authors.

Now here is a comparative analysis result of various object detection methods.

COMPARATIVE RESULTS ON VOC 2007 TEST SET (%).

Methods	Trained on	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN (Alex) [15]	07	68.1	72.8	56.8	43.0	36.8	66.3	74.2	67.6	34.4	63.5	54.5	61.2	69.1	68.6	58.7	33.4	62.9	51.1	62.5	68.6	58.5
R-CNN(VGG16) [15]	07	73.4	77.0	63.4	45.4	44.6	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	35.6	66.8	67.2	70.4	71.1	66.0
SPP-net(ZF) [64]	07	68.5	71.7	58.7	41.9	42.5	67.7	72.1	73.8	34.7	67.0	63.4	66.0	72.5	71.3	58.9	32.8	60.9	56.1	67.9	68.8	60.9
GCNN [70]	07	68.3	77.3	68.5	52.4	38.6	78.5	79.5	81.0	47.1	73.6	64.5	77.2	80.5	75.8	66.6	34.3	65.2	64.4	75.6	66.4	66.8
Bayes [85]	07	74.1	83.2	67.0	50.8	51.6	76.2	81.4	77.2	48.1	78.9	65.6	77.3	78.4	75.1	70.1	41.4	69.6	60.8	70.2	73.7	68.5
Fast R-CNN [16]	07+12	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4	70.0
SDP+CRC [33]	07	76.1	79.4	68.2	52.6	46.0	78.4	78.4	81.0	46.7	73.5	65.3	78.6	81.0	76.7	77.3	39.0	65.1	67.2	77.5	70.3	68.9
SubCNN [60]	07	70.2	80.5	69.5	60.3	47.9	79.0	78.7	84.2	48.5	73.9	63.0	82.7	80.6	76.0	70.2	38.2	62.4	67.7	77.7	60.5	68.5
StuffNet30 [100]	07	72.6	81.7	70.6	60.5	53.0	81.5	83.7	83.9	52.2	78.9	70.7	85.0	85.7	77.0	78.7	42.2	73.6	69.2	79.2	73.8	72.7
NOC [114]	07+12	76.3	81.4	74.4	61.7	60.8	84.7	78.2	82.9	53.0	79.2	69.2	83.2	83.2	78.5	68.0	45.0	71.6	76.7	82.2	75.7	73.3
MR-CNN&S-CNN [110]	07+12	80.3	84.1	78.5	70.8	**68.5**	88.0	85.9	87.8	60.3	85.2	73.7	87.2	86.5	85.0	76.4	48.5	76.3	75.5	85.0	81.0	78.2
HyperNet [101]	07+12	77.4	83.3	75.0	69.1	62.4	83.1	87.4	87.4	57.1	79.8	71.4	85.1	85.1	80.0	79.1	51.2	79.1	75.7	80.9	76.5	76.3
MS-GR [104]	07+12	80.0	81.0	77.4	72.1	64.3	88.2	88.1	88.4	64.4	85.4	73.1	87.3	87.4	85.1	79.6	50.1	78.4	79.5	86.9	75.5	78.6
OHEM+Fast R-CNN [113]	07+12	80.6	85.7	79.8	69.9	60.8	88.3	87.9	89.6	59.7	85.1	76.5	87.1	87.3	82.4	78.8	53.7	80.5	78.7	84.5	80.7	78.9
ION [95]	07+12+S	80.2	85.2	78.8	70.9	62.6	86.6	86.9	**89.8**	61.7	86.9	76.5	**88.4**	87.5	83.4	80.5	52.4	78.1	77.2	86.9	**83.5**	79.2
Faster R-CNN [18]	07	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6	69.9
Faster R-CNN [18]	07+12	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6	73.2
Faster R-CNN [18]	07+12+COCO	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9	78.8
SSD300 [71]	07+12+COCO	80.9	86.3	79.0	**76.2**	57.6	87.3	88.2	88.6	60.5	85.4	**76.7**	87.5	**89.2**	84.5	81.4	55.0	81.9	**81.5**	85.9	78.9	79.6
SSD512 [71]	07+12+COCO	**86.6**	**88.3**	**82.4**	76.0	66.3	**88.6**	**88.9**	89.1	**65.1**	**88.4**	73.6	86.5	88.9	**85.3**	**84.6**	**59.1**	**85.0**	80.4	**87.4**	81.2	**81.6**

* '07': VOC2007 trainval, '07+12': union of VOC2007 and VOC2012 trainval, '07+12+COCO': trained on COCO trainval35k at first and then fine-tuned on 07+12. The S in ION '07+12+S' denotes SBD segmentation labels.

Comparative Result of various methods

COMPARATIVE RESULTS ON VOC 2012 TEST SET (%).

Methods	Trained on	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN(Alex) [15]	12	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1	53.3
R-CNN(VGG16) [15]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
Bayes [85]	12	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2	66.4
Fast R-CNN [16]	07++12	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2	68.4
SutffNet30 [100]	12	83.0	76.9	71.2	51.6	50.1	76.4	75.7	87.8	48.3	74.8	55.7	85.7	81.2	80.3	79.5	44.2	71.8	61.0	78.5	65.4	70.0
NOC [114]	07+12	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1	68.8
MR-CNN&S-CNN [110]	07++12	85.5	82.9	76.6	57.8	62.7	79.4	77.2	86.6	55.0	79.1	62.2	87.0	83.4	84.7	78.9	45.3	73.4	65.8	80.3	74.0	73.9
HyperNet [101]	07++12	84.2	78.5	73.6	55.6	53.7	78.7	79.8	87.7	49.6	74.9	52.1	86.0	81.7	83.3	81.8	48.6	73.5	59.4	79.9	65.7	71.4
OHEM+Fast R-CNN [113]	07++12+coco	90.1	87.4	79.9	65.8	66.3	86.1	85.0	92.9	62.4	83.4	69.5	90.6	88.9	88.9	83.6	59.0	82.0	74.7	88.2	77.3	80.1
ION [95]	07+12+S	87.5	84.7	76.8	63.8	58.3	82.6	79.0	90.9	57.8	82.0	64.7	88.9	86.5	84.7	82.3	51.4	78.2	69.2	85.2	73.5	76.4
Faster R-CNN [18]	07++12	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5	70.4
Faster R-CNN [18]	07++12+coco	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2	75.9
YOLO [17]	07++12	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8	57.9
YOLO+Fast R-CNN [17]	07++12	83.4	78.5	73.5	55.8	43.4	79.1	73.1	89.4	49.4	75.5	57.0	87.5	80.9	81.0	74.7	41.8	71.5	68.5	82.1	67.2	70.7
YOLOv2 [72]	07++12+coco	88.8	87.0	77.8	64.9	51.8	85.2	79.3	93.1	64.4	81.4	**70.2**	91.3	88.1	87.2	81.0	57.7	78.1	71.0	88.5	76.8	78.2
SSD300 [71]	07++12+coco	91.0	86.0	78.1	65.0	55.4	84.9	84.0	93.4	62.1	83.6	67.3	91.3	88.9	88.6	85.6	54.7	83.8	77.3	88.3	76.5	79.3
SSD512 [71]	07++12+coco	91.4	88.6	82.6	71.4	63.1	**87.4**	88.1	93.9	66.9	86.6	66.3	92.0	91.7	90.8	88.5	60.9	87.0	75.4	90.2	80.4	82.2
R-FCN (ResNet101) [16]	07++12+coco	**92.3**	**89.9**	**86.7**	**74.7**	**75.2**	86.7	**89.0**	**95.8**	**70.2**	**90.4**	66.5	**95.0**	**93.2**	**92.1**	**91.1**	**71.0**	**89.7**	**76.0**	**92.0**	**83.4**	**85.0**

* '07++12': union of VOC2007 trainval and test and VOC2012 trainval. '07++12+COCO': trained on COCO trainval35k at first then fine-tuned on 07++12.

COMPARATIVE RESULTS ON MICROSOFT COCO TEST DEV SET

COMPARATIVE RESULTS ON MICROSOFT COCO TEST DEV SET (%).

Methods	Trained on	0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [16]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.4	30.1	7.3	32.1	52.0
ION [95]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
NOC+FRCN(VGG16) [114]	train	21.2	41.5	19.7	-	-	-	-	-	-	-	-	-
NOC+FRCN(Google) [114]	train	24.8	44.4	25.2	-	-	-	-	-	-	-	-	-
NOC+FRCN (ResNet101) [114]	train	27.2	48.4	27.6	-	-	-	-	-	-	-	-	-
GBD-Net [109]	train	27.0	45.8	-	-	-	-	-	-	-	-	-	-
OHEM+FRCN [113]	train	22.6	42.5	22.2	5.0	23.7	34.6	-	-	-	-	-	-
OHEM+FRCN* [113]	train	24.4	44.4	24.8	7.1	26.4	37.9	-	-	-	-	-	-
OHEM+FRCN* [113]	trainval	25.5	45.9	26.1	7.4	27.7	38.5	-	-	-	-	-	-
Faster R-CNN [18]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
YOLOv2 [72]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4
SSD300 [71]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [71]	trainval35k	26.8	46.5	27.8	9.0	28.9	41.9	24.8	37.5	39.8	14.0	43.5	59.0
R-FCN (ResNet101) [65]	trainval	29.2	51.5	-	10.8	32.8	45.0	-	-	-	-	-	-
R-FCN*(ResNet101) [65]	trainval	29.9	51.9	-	10.4	32.4	43.3	-	-	-	-	-	-
R-FCN**(ResNet101) [65]	trainval	31.5	53.2	-	14.3	35.5	44.2	-	-	-	-	-	-
Multi-path [112]	trainval	33.2	51.9	36.3	13.6	37.2	47.8	29.9	46.0	48.3	23.4	56.0	66.4
FPN (ResNet101) [66]	trainval35k	36.2	59.1	39.0	18.2	39.0	48.2	-	-	-	-	-	-
Mask (ResNet101+FPN) [67]	trainval35k	38.2	60.3	41.7	20.1	41.1	50.2	-	-	-	-	-	-
Mask (ResNeXt101+FPN) [67]	trainval35k	**39.8**	**62.3**	**43.4**	**22.1**	**43.2**	**51.2**	-	-	-	-	-	-
DSSD513 (ResNet101) [73]	trainval35k	33.2	53.3	35.2	13.0	35.4	51.1	28.9	43.5	46.2	21.8	49.1	66.4
DSOD300 [74]	trainval	29.3	47.3	30.6	9.4	31.5	47.0	27.3	40.7	43.0	16.7	47.1	65.0

* FRCN*: Fast R-CNN with multi-scale training, R-FCN*: R-FCN with multi-scale training, R-FCN**: R-FCN with multi-scale training and testing, Mask: Mask R-CNN.

Comparative Result on V07 dataset

COMPARISON OF TESTING CONSUMPTION ON VOC 07 TEST SET.

Methods	Trained on	mAP(%)	Test time(sec/img)	Rate(FPS)
SS+R-CNN [15]	07	66.0	32.84	0.03
SS+SPP-net [64]	07	63.1	2.3	0.44
SS+FRCN [16]	07+12	66.9	1.72	0.6
SDP+CRC [33]	07	68.9	0.47	2.1
SS+HyperNet* [101]	07+12	76.3	0.20	5
MR-CNN&S-CNN [110]	07+12	78.2	30	0.03
ION [95]	07+12+S	79.2	1.92	0.5
Faster R-CNN(VGG16) [18]	07+12	73.2	0.11	9.1
Faster R-CNN(ResNet101) [18]	07+12	**83.8**	2.24	0.4
YOLO [17]	07+12	63.4	**0.02**	**45**
SSD300 [71]	07+12	74.3	**0.02**	**46**
SSD512 [71]	07+12	76.8	0.05	19
R-FCN(ResNet101) [65]	07+12+coco	83.6	0.17	5.9
YOLOv2(544*544) [72]	07+12	78.6	0.03	40
DSSD321(ResNet101) [73]	07+12	78.6	0.07	13.6
DSOD300 [74]	07+12+coco	81.7	0.06	17.4
PVANET+ [116]	07+12+coco	**83.8**	0.05	21.7
PVANET+(compress) [116]	07+12+coco	82.9	0.03	31.3

* SS: Selective Search [15], SS*: 'fast mode' Selective Search [16], HyperNet*: the speed up version of HyperNet and PAVNET+ (compresss): PAVNET with additional bounding box voting and compressed fully convolutional layers.

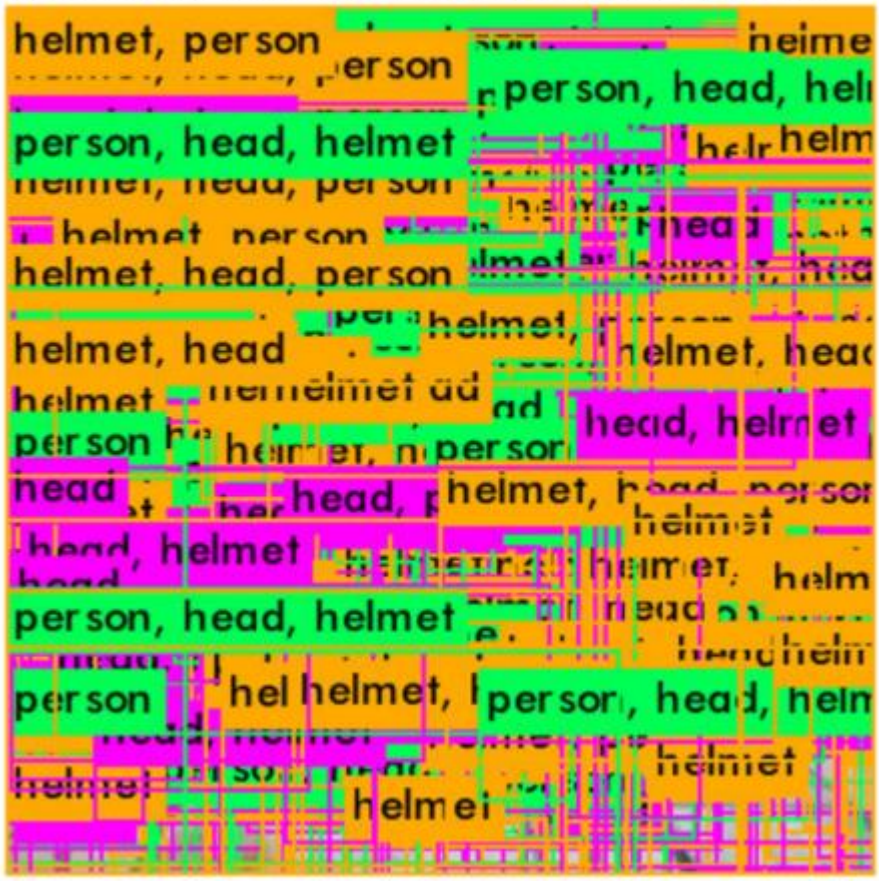
Performance Comparison of YOLO

Network Size	Darknet, FPS (avg)	tkDNN TensorRT FP32, FPS	tkDNN TensorRT FP16, FPS	OpenCV FP16, FPS	tkDNN TensorRT FP16 batch=4, FPS	OpenCV FP16 batch=4, FPS	tkDNN Speedup
320	100	116	**202**	183	423	**430**	**4.3x**
416	82	103	**162**	159	284	**294**	**3.6x**
512	69	91	134	**138**	206	**216**	**3.1x**
608	53	62	103	**115**	150	**150**	**2.8x**
Tiny 416	443	609	**790**	773	**1774**	1353	**3.5x**
Tiny 416 CPU Core i7 7700HQ	3.4	-	-	42	-	39	**12x**

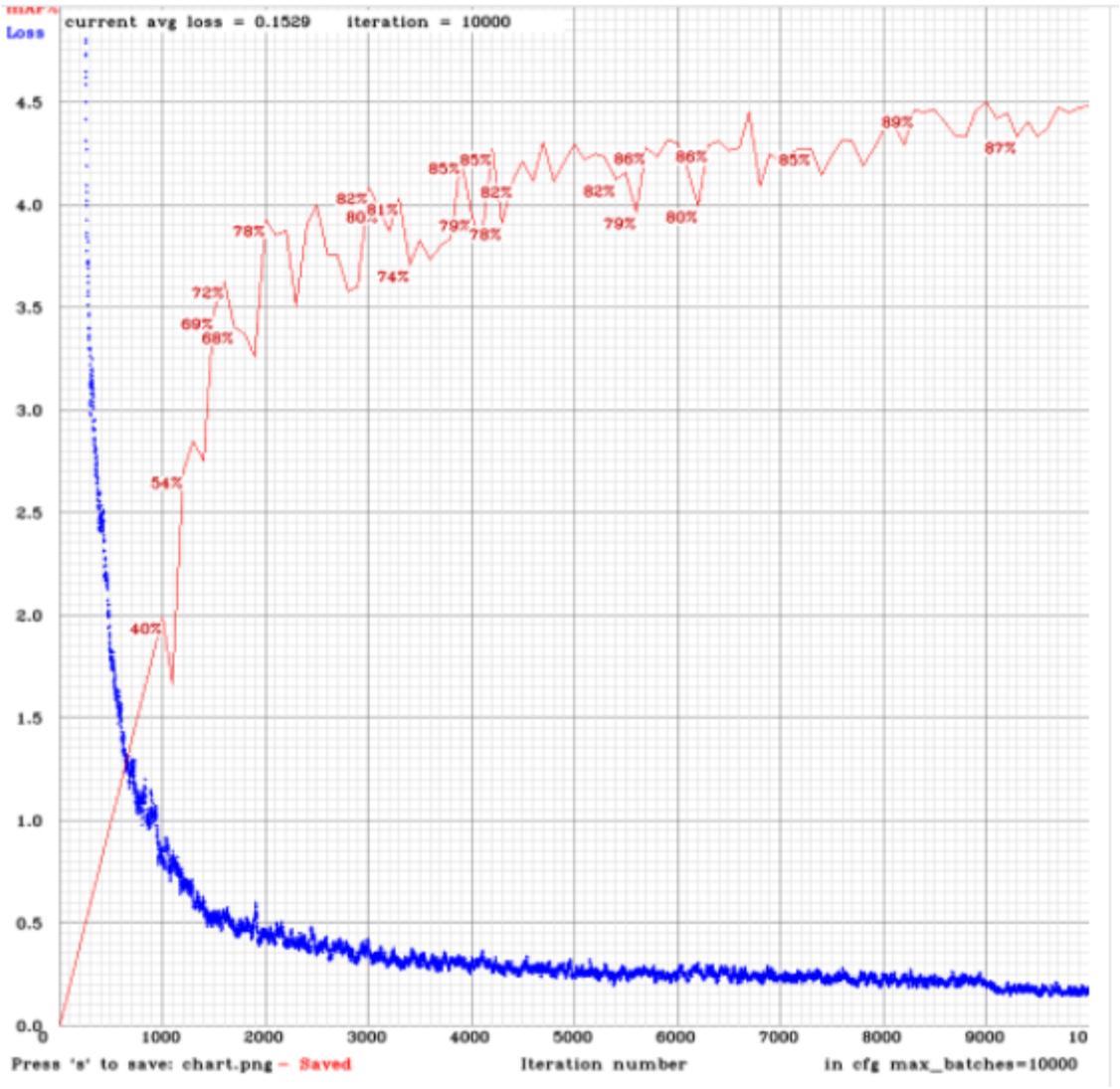
Here are result of the model after 100 iterations

```
helmet: 27%
helmet: 35%
head: 27%
helmet: 25%
Unable to init server: Could not connect: Connection refused

(predictions:2768): Gtk-WARNING **: 01:53:00.375: cannot open display:
```



Below is the mAP and Loss Chart for the following model



Object detection is a core problem in computer vision. Pipeline of Detection generally start by robust feature extraction from input media (Haar , SIFT , HOG , convolutional features). At that point, classifiers or localizers are utilized to distinguish objects in the feature space. These classifiers or localizers are run either in sliding window style over the entire picture or on some subset of regions in the picture. When comparing YOLO in contrast to modern object detection systems, featuring similarities and differences. Deformable parts models. Deformable parts models (DPM) utilize a sliding window approach to object detection . DPM utilizes a disjoint pipeline to extract static features, group regions, predict bounding boxes for high scoring regions, and so forth Our system replaces these disparate parts with a solitary convolutional neural network. The network

performs feature extraction, bounding box prediction, nonmaximal suppression, and relevant reasoning all concurrently. Rather than static features, the network trains the features in-line and streamlines them for the detection task. Our brought together architecture prompts a faster, more accurate model than DPM. R-CNN. R-CNN and its variants use region proposals as opposed to sliding windows to discover objects in pictures. Specific 4 Search generates potential bounding boxes, a convolutional network extracts features, a SVM scores the boxes, a linear model changes the bounding boxes, and non-max suppression wipes out copy detections. Each phase of this mind-boggling pipeline must be precisely tuned freely and the resulting system is very moderate, taking more than 40 seconds per picture at test time . YOLO shares a few similarities with R-CNN. Every grid cell proposes a potential bounding box and scores those boxes utilizing convolutional features. However, our system puts spatial constraints on the grid cell proposals which mitigates different detections of a similar object. Our system likewise proposes far fewer bounding boxes, just 98 per picture compared to around 2000 from Selective Search. At last, our system consolidates these individual segments into a solitary, together upgraded model. Other Fast Detectors Fast and Faster R-CNN center around accelerating the R-CNN framework by sharing calculation and utilizing neural networks to propose regions rather than Selective Search. While they offer speed and accuracy improvements over R-CNN, both still miss the mark regarding real-time performance. Many research efforts center around accelerating the DPM pipeline. They accelerate HOG calculation, use falls, and push calculation to GPUs. However, just 30Hz DPM really runs in real-time. Rather than trying to improve singular parts of a large detection pipeline, YOLO throws out the pipeline entirely and is quick by plan. Detectors for single classes like faces or individuals can be profoundly enhanced since they need to manage significantly less variation. YOLO is a general-purpose detector that learns to identify a variety of objects all the while. Profound MultiBox. Dissimilar to R-CNN, Szegedy et al. train a convolutional neural network to predict regions of interest as opposed to utilizing Selective Search.

MultiBox can likewise perform single object detection by replacing the certainty prediction with a solitary class prediction. However, MultiBox can't perform general object detection is still a piece in a larger detection pipeline, requiring further picture fix characterization. Both YOLO and MultiBox utilize a convolutional network to predict bounding boxes in a picture however YOLO is a finished

detection system. OverFeat. Sermanet et al. train and test a CNN to perform normalization and adjust the localizer to do detection . OverFeat effectively performs sliding window detection yet it is as yet a disjoint system. OverFeat improves for limitation, not detection performance. Like DPM, the localizer possibly observes neighborhood information when making a prediction. OverFeat can't reason about worldwide setting and consequently requires critical post-processing to produce coherent detections. MultiGrasp. Our work is similar in plan to work on grasp detection by Redmon et al . Our grid approach to bounding box prediction depends on the MultiGrasp system for regression to grasps. However, grasp detection is a lot simpler undertaking than object detection. MultiGrasp just requirements to predict a solitary graspable region for a picture containing one object. It doesn't need to assess the size, area or boundaries of the object or predict its class just discover a region reasonable for gripping. YOLO predicts both bounding boxes and class probabilities for various objects of different classes in a picture. Experiments First we compare YOLO with other real-time detection systems on PASCAL VOC 2007. Key contrast among YOLO and R-CNN variants are to explore the errors on VOC 2007 made by YOLO and Fast R-CNN, one of the most efficient versions of R-CNN . In view of the different error profiles we show that YOLO can be utilized to rescore Fast R-CNN detections and reduce the errors from background bogus positives, giving a huge performance help. We additionally present VOC 2012 results and compare mAP to current best in class strategies. At last, we show that YOLO generalizes to new spaces better than other detectors on two artwork datasets.

Drawbacks and Contribution

The first one is little object detection, for example, occurring in COCO dataset and in face detection task. To improve limitation accuracy on little objects under partial impediments, it is necessary to change network architectures from the accompanying angles.

• Multi-task joint enhancement and multi-modular information combination. Because of the correlations between different assignments inside and outside object detection, perform multiple tasks joint enhancement has already been concentrated by numerous researchers. However, apart from the errands

referenced in Subs. III-A8, it is desirable to thoroughly consider the characteristics of different sub-assignments of object detection (for example superpixel semantic division in notable object detection) and stretch out perform various tasks advancement to other applications, for example, case division [66], multi-object tracking and multi-person present assessment . Additionally, given a particular application, the information from different modalities, for example, text , thermal information and pictures , can be combined to accomplish a more discriminant network.

• Scale adaption. Objects for the most part exist in different scales, which is more apparent in face detection and pedestrian detection. To increase the robustness to scale transforms, it is requested to train scale-invariant, multi-scale or scaleadaptive detectors. For scale-invariant detectors, more powerful spine architectures (for example ResNext), negative example mining , reverse association and subcategory modeling are altogether useful. For multi-scale detectors, both the FPN which produces multi-scale feature maps and Generative Adversarial Network which narrows representation differences between little objects and the large ones with an ease architecture provide experiences into generating significant feature pyramid. For scale-versatile detectors, it is valuable to consolidate information graph, attentional component , course network and scale distribution assessment to distinguish objects adaptively.

• Spatial correlations and logical modeling. Spatial distribution assumes an important role in object detection. So, region proposal generation and grid regression are taken to get probable object areas. However, the correlations between numerous proposals and object categories are ignored. Plus, the worldwide structure information is deserted by the position-delicate score maps in R-FCN. To tackle these problems, we can refer to diverse subset choice and successive reasoning errands for potential arrangements. It is additionally important to veil notable parts and couple them with the worldwide structure in a joint-learning manner. The subsequent one is to release the burden on physical work and achieve real-time object detection, with the emergence of large-scale picture and video information. The accompanying three angles can be considered.
• Cascade network. In a course network, a course of detectors are implicit different stages or layers. Furthermore, effectively recognizable models are rejected at shallow layers so that features and classifiers at latter stages can deal

with more troublesome examples with the guide of the choices from previous stages. However, current falls are inherent a greedy manner, where previous stages in course are fixed when training another stage. So, the improvements of different CNNs are disconnected, which stresses the need of end-to end advancement for CNN course. Simultaneously, it is additionally a matter of concern to construct logical related course networks with existing layers.

• Unsupervised and pitifully supervised learning. It's very tedious to physically draw large amounts of bounding boxes. To release this burden, semantic prior, unsupervised object discovery, numerous occurrences learning and profound neural network prediction can be integrated to utilize picture level supervision to allot object category labels to corresponding object regions and refine object boundaries. Furthermore, pitifully comments (for example centre-click comments) are additionally useful for accomplishing great detectors with unassuming comment efforts, particularly supported by the portable platform.

• Network enhancement. Given explicit applications and platforms, it is critical to make an equilibrium among speed.

Further on we study the impact of different spine models on the detector accuracy. We notice that the model characterized with the best grouping accuracy isn't generally the best in terms of the detector accuracy. First, in spite of the fact that grouping accuracy of CSPResNeXt50 models trained with different features is higher compared to CSPDarknet53 models, the CSPDarknet53 model shows higher accuracy in terms of object detection. Second, utilizing BoF and Mish for the CSPResNeXt50 classifier training increases its grouping accuracy, however further utilization of these pre-trained weightings for detector training reduces the detector accuracy. However, utilizing BoF and Mish for the CSPDarknet53 classifier training increases the accuracy of both the classifier and the detector which utilizes this classifier pre-trained weightings. The performance of CSPDarknet53 is better in terms of computational load than CSPResNeXt50. We observe that the CSPDarknet53 model demonstrates a greater capacity to increase the detector accuracy inferable from various improvements.

Innovation

From YOLO v1 to v2 following changes were made-

Bunch Normalization: it normalizes the information layer by altering somewhat and scaling the enactments. Cluster normalization decreases the move in unit esteem in the shrouded layer and thusly it improves the soundness of the neural network. By adding clump normalization to convolutional layers in the architecture MAP (mean average precision) has been improved by 2%. It likewise helped the model regularize and overfitting has been reduced overall.

Higher Resolution Classifier: the scan size in YOLO v2 has been incremented from 224x224 to 448x448. The increase in the information size of the picture has improved the MAP (mean average precision) up to 4%. This increase in info size is been applied while training the YOLO v2 architecture DarkNet 19 on ImageNet dataset.

Anchor Boxes: major noticeable change in YOLO is the inclusion of the anchor boxes. YOLO v2 does characterization and prediction in a solitary framework. These anchor boxes are responsible for predicting bounding box and this anchor boxes are intended for a given dataset by utilizing clustering (k-implies clustering).

Fine-Grained Features: one of the fundamentals gave that must be addressed in the YOLO v1 is that detection of smaller objects on the picture. This has been resolved in the YOLO v2 partitions the picture into 13*13 grid cells which is smaller when compared to its previous version. This empowers the yolo v2 to distinguish or confine the smaller objects in the picture and furthermore successful with the larger objects.

Multi-Scale Training: on YOLO v1 has a shortcoming recognizing objects with different information sizes which says that if YOLO is trained with little pictures of

a particular object it has issues distinguishing a similar object on picture of bigger size. This has been resolved to a great degree in YOLO v2 where it is trained with random pictures with different measurements range between 320x320 to 608x608 . This permits the network to learn and predict the objects from various info measurements with accuracy.

Darknet 19: YOLO utilizes Darknet architecture with 19 layers of convolutional network and 5 pooling layers and a softmax layer for identifying objects. The architecture of the Darknet 19 has been demonstrated as follows. Darknet is a neural network framework written in C language and CUDA. It's really quick in object detection which is very important for predicting in real-time.

Also, from YOLOv2 to v 4 after changes are incurred-

Bounding Box Predictions: In YOLO v3 gives the score for the objects for each bounding boxes. It utilizes calculated regression to predict the objectiveness score.

Class Predictions: In YOLO v3 it utilizes strategic classifiers for every class rather than softmax which has been utilized in the previous YOLO v2. By doing as such in YOLO v3 we can have multi-mark order. With softmax layer if the network is trained for both a person and man, it gives the probability among person and man suppose 0.4 and 0.47. With the autonomous classifier gives the probability for each class of objects. For instance if the network is trained for person and a man it would give the probability of 0.85 to person and 0.8 for the man and mark the model objects as person.

Feature Pyramid Networks (FPN): YOLO v3 makes predictions similar to the FPN where 3 predictions are made for every area the information picture and features are extracted from every prediction. By doing so YOLO v3 has the better capacity at different scales. As clarified from the paper by every prediction is made with

boundary box, objectless and 80 class scores. Doing up sampling from previous layers permits getting importance full semantic information and finer-grained information from earlier feature map. Presently, adding not many more convolutional layers to process improves the yield .

Darknet-53: the predecessor YOLO v2 utilized Darknet-19 as feature extractor and YOLO v3 utilizes the Darknet-53 network for feature extractor which has 53 convolutional layers. It is a lot deeper than the YOL v2 and furthermore had shortcut associations. . Darknet-53 makes out of the predominantly with 3x3 and 1x1 filters with shortcut associations.

Conclusion

We introduce YOLO, a bound together model for object detection. Our model is easy to construct and can be trained directly on full pictures. Not at all like classifier-based approaches, YOLO is trained on a misfortune work that directly corresponds to detection performance and the entire model is trained mutually. Quick YOLO is the quickest general-purpose object detector in the literature and YOLO pushes the cutting edge in real-time object detection. YOLO additionally generalizes well to new areas making it ideal for applications that rely on quick, robust object detection

We offer a cutting-edge detector which is faster (FPS) and more accurate (MS COCO AP50...95 and AP50) than all accessible alternative detectors. The detector described can be trained and utilized on an ordinary GPU with 8-16 GB-VRAM this makes its broad utilize conceivable. The original idea of one-stage anchor-based detectors has proven its reasonability. We have verified a large number of features, and chose for utilize such of them for improving the accuracy of both the classifier and the detector. These features can be utilized as best-practice for future examinations and advancements.

Because of its powerful learning capacity and favourable circumstances in managing impediment, scale transformation and background switches, profound learning-based object detection has been a research hotspot in recent years. This paper provides a point by point review on profound learning-based object detection frameworks which handle different sub-problems, for example,

impediment, clutter and low resolution, with different degrees of alterations on R-CNN. The review starts on generic object detection pipelines which provide base architectures for other related errands. At that point, three other normal errands, to be specific remarkable object detection, face detection and pedestrian detection, are additionally briefly reviewed. At last, we propose several promising future directions to increase a thorough understanding of the object detection scene. This review is additionally significant for the improvements in neural networks and related learning systems, which provides important bits of knowledge and rules for future progress.

Reference

- YOLOv4: Optimal Speed and Accuracy of Object Detection(Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark)
- You Only Look Once: Unified, Real-Time Object Detection(Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi)
- Object Detection with Deep Learning: A Review(Zhong-Qiu Zhao, Shou-tao Xu, Xindong Wu)
- Soft-NMS–improving object detection with one line of code. IEEE International Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis Conference on Computer Vision (ICCV)
- Cascade R-CNN: Delving into high quality object detection. Zhaowei Cai and Nuno Vasconcelos. IEEE Conference on Computer Vision and Pattern
- Hierarchical shot detector. Jiale Cao, Yanwei Pang, Jungong Han, and Xuelong Li. IEEE International Conference on Computer Vision (ICCV)
- HarDNet: A low memory traffic network. Huang, and Youn-Long Lin. IEEE International Conference on Computer Vision (ICCV),
- Learning to localize objects with structured output regression. M. B. Blaschko and C. H. Lampert. In Computer Vision– ECCV
- Body part detectors trained using 3d human pose annotations. L. Bourdev and J. Malik. Poselets. In International Conference on Computer Vision (ICCV), 2009
- The crossdepiction problem: Computer vision algorithms for recognising objects in artwork and in photographs, H. Cai, Q. Wu, T. Corradi, and P. Hall. 2015.