Aim:
To build a model that can detect cars and people in any given image. The model should be trained using the given dataset. The dataset follows coco annotations and has 2238 images.

Business Use case:
We can use this approach to train models for the identification of different items.

Methodology:

1) Dataset Annotation Conversion. - The dataset which was given had coco annotations. Such annotations cannot be used as an input to Yolov4 or Yolov5. They both had different input dataset formats. The conversion.py file in GitHub shows how annotations were converted.
2) To evaluate pre-trained models on the test dataset.
3) To use Transfer Learning,i.e to use pre-trained models, to train on our dataset and further compare them. This will usually be the core information for the model to function, with new aspects added to the model to solve a specific task. Programmers will need to identify which areas of the model are relevant to the new task, and which parts will need to be retrained. For example, a new model may keep the processes that allow the machine to identify objects or data, but retrain the model to identify a different specific object.
   Benefits of Transfer learning:

- Removing the need for a large set of labelled training data for every new model.
- Improving the efficiency of machine learning development and deployment for multiple models.
- A more generalised approach to machine problem solving, leveraging different algorithms to solve new challenges.
- Models can be trained within simulations instead of real-world environments.

4) And atlast, we will evaluate all models and compare their performance on basis of recall, precision and Mean Average Precision.

Models used:

1) YOLOV4:: The first version of YOLO was introduced by Joseph Redmon and his co-authors in 2015 which made a breakthrough in real-time object detection. YOLOv1 is a one-stage object detector with fast inference speed and acceptable accuracy compared with two-stage methods at that time. YOLOv2, also referred to as YOLO9000, was proposed one year later to improve the detection accuracy by applying the concept of anchor box. In 2016, further improvements were provided in YOLOv3 with a new backbone network Darknet53 and the capability of detecting objects at three different scales using Feature Pyramid Network (FPN) as the model neck. From the next version, YOLOv4, Joseph announced that he stopped going on this project due to some individual reasons and gave the leading privilege of the YOLO project to Alexey Bochkovskiy, and Alexey introduced YOLOv4 in 2020. YOLOv4 has improved the performance of the predecessor YOLOv3 by using a new backbone, CSPDarknet53

(CSP stands for Cross Stage Partial), adding Spatial Pyramid Pooling (SPP), Path Aggregation Network (PAN), and introducing mosaic data augmentation method.

2) YOLOV5: YOLOv5 is a family of object detection architectures and models pretrained on the COCO dataset, and represents Ultralytics open-source research into future vision AI methods, incorporating lessons learned and best practices evolved over thousands of hours of research and development.

Discussion

A) Why two models?
Comparing two pre-trained models will help us achieve a better understanding and it will be easy to compare overall results.

B) Why Yolov4?
In experiments, YOLOv4 obtained an AP value of 43.5 percent (65.7 percent AP50) on the MS COCO dataset, and achieved a real-time speed of ~65 FPS on the Tesla V100, beating the fastest and most accurate detectors in terms of both speed and accuracy. YOLOv4 is twice as fast as EfficientDet with comparable performance. In addition, compared with YOLOv3, the AP and FPS have increased by 10 percent and 12 percent, respectively. Some of the features of YoloV4 are:

- Cross-Stage-Partial-Connections (CSP), A new backbone that can enhance learning capability of CNN
- Cross mini-Batch Normalization (CmBN), represents a CBN modified version which assumes a batch contains four mini-batches
- Self-adversarial-training (SAT), represents a new data augmentation technique that operates in 2 forward backward stages
- Mish-activation, A novel self regularized non-monotonic neural activation function
- Mosaic data augmentation, represents a new data augmentation method that mixes 4 training images instead of a single image
- DropBlock regularization, a better regularization method for CNN
- CIoU loss, achieves better convergence speed and accuracy on the BBox regression problem.

C) Why Yolov5?

- YOLOv5 is implemented in PyTorch initially, it benefits from the established PyTorch ecosystem: support is simpler, and deployment is easier. Moreover as a more widely known research framework, iterating on YOLOv5 may be easier for the broader research community
- YOLOv5 is fast
- YOLOv5 is accurate. In tests on the blood cell count and detection (BCCD) dataset, it achieved roughly 0.895 mean average precision (mAP) after training for just 100 epochs. Admittedly, we saw comparable performance from

EfficientDet and YOLOv4, but it is rare to see such across-the-board performance improvements without any loss in accuracy.

- YOLOv5 is nearly 90 percent smaller than YOLOv4. This means YOLOv5 can be deployed to embedded devices much more easily.

D) Why not other models?

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
CenterNet MobileNetV2 FPN 512x512	6	23.4	Boxes
CenterNet MobileNetV2 FPN Keypoints 512x512	6	41.7	Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes
EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes
EfficientDet D7 1536x1536	325	51.2	Boxes
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3	Boxes
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38.3	Boxes
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29.3	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31.0	Boxes
Faster R-CNN ResNet50 V1 800x1333	65	31.6	Boxes
Faster R-CNN ResNet101 V1 640x640	55	31.8	Boxes
Faster R-CNN ResNet101 V1 1024x1024	72	37.1	Boxes
Faster R-CNN ResNet101 V1 800x1333	77	36.6	Boxes
Faster R-CNN ResNet152 V1 640x640	64	32.4	Boxes
Faster R-CNN ResNet152 V1 1024x1024	85	37.6	Boxes
Faster R-CNN ResNet152 V1 800x1333	101	37.4	Boxes
Faster R-CNN Inception ResNet V2 640x640	206	37.7	Boxes
Faster R-CNN Inception ResNet V2 1024x1024	236	38.7	Boxes
Mask R-CNN Inception ResNet V2 1024x1024	301	39.0/34.6	Boxes/Masks
ExtremeNet (deprecated)	--	--	Boxes
ExtremeNet	--	--	Boxes

The above table compares the performance of various models using the coco 2017 dataset.

Problems Faced:

1) Difficulty with dataset annotation - Since it was in coco we had to change it, and since it was in JSON format we had to parse the file thoroughly and then to get individual file.
2) As we were using Google Colab, we faced difficulties in training on GPU since it gave limited time period access.
3) Whenever, yolov5 takes input final, so in custom yaml file we have to use class 0,1 for categories rather than 1,2; i.e in our dataset we had two classes 1,2 but yolov5 takes input of category as 0,1 for 2 classes, so we had to change it.

Results:

We weren't able to compare the performance of our model as training was continuously getting crashed and thus it affected time management. (computational limitation).