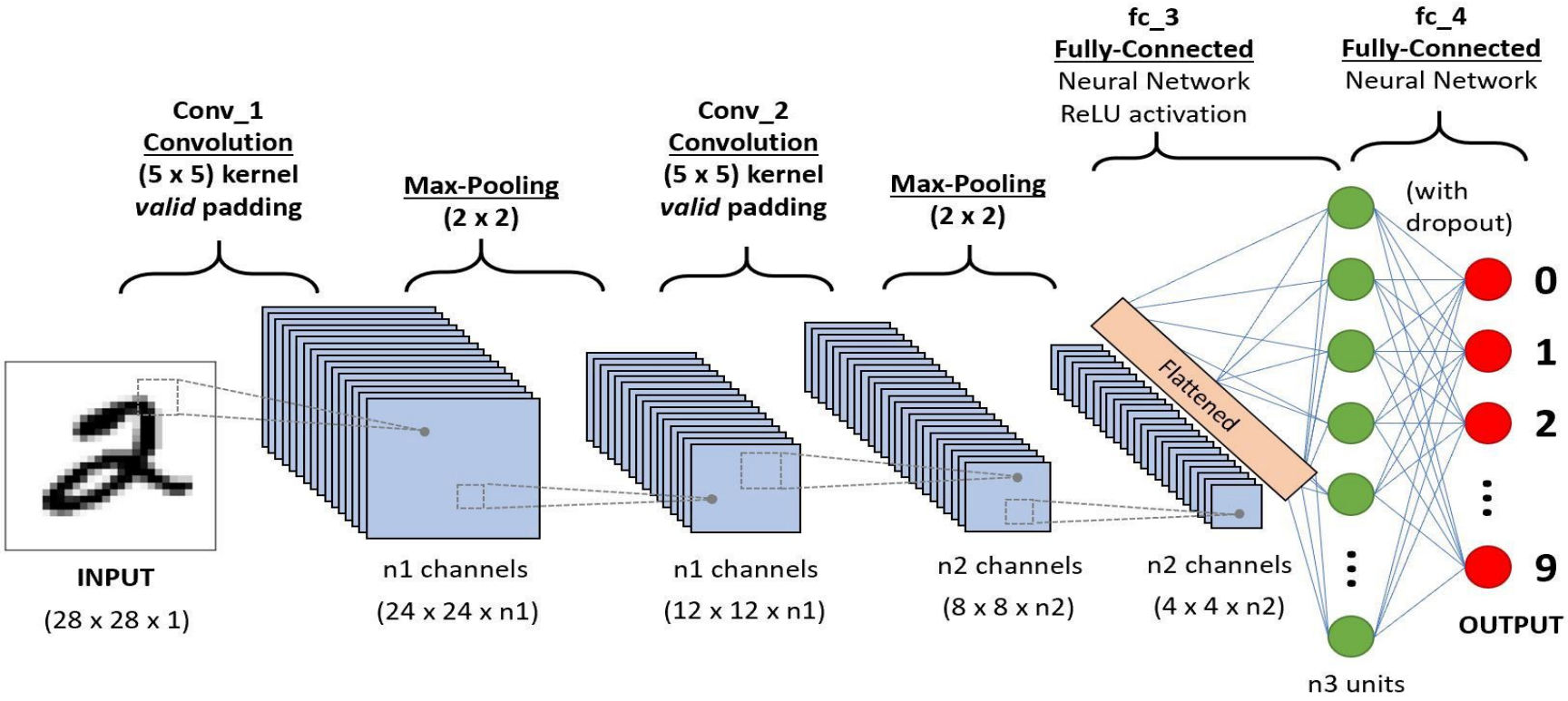
Practical Object Detection

Shreeyash Pawar

link:https://github.com/shree970/Real-Time-Object-Detection

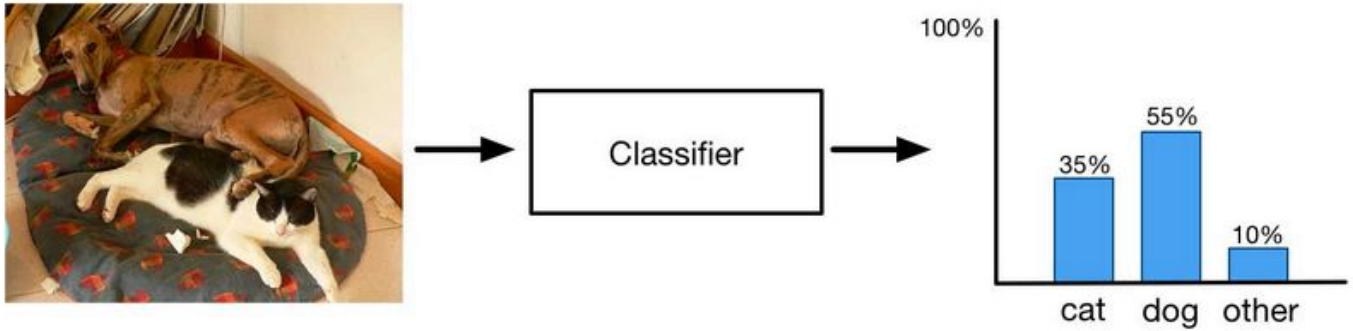
CNN: Recap

Because no CV talk starts without it



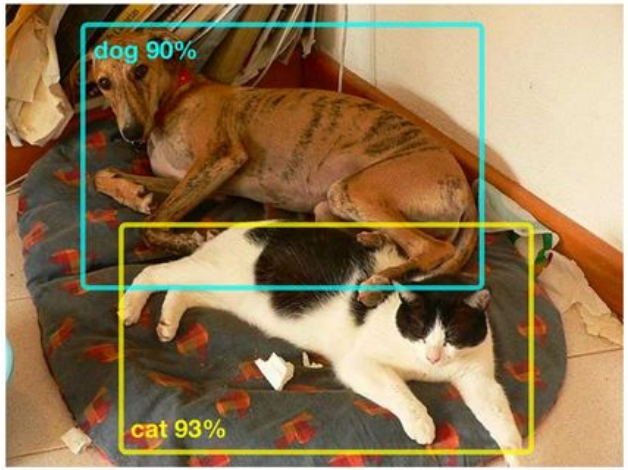
What is Object Detection?

Instead of this:



We require this:

localisation + classification



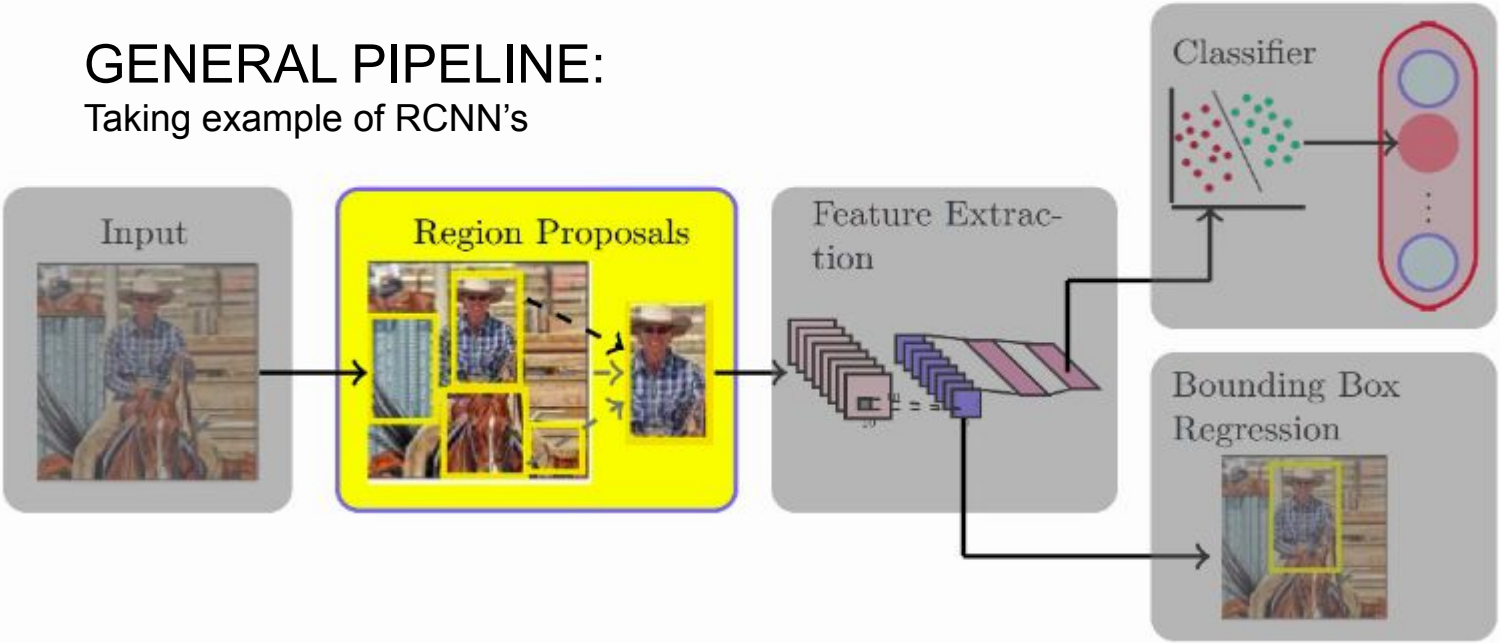
Observe: we always have to deal with two functions:

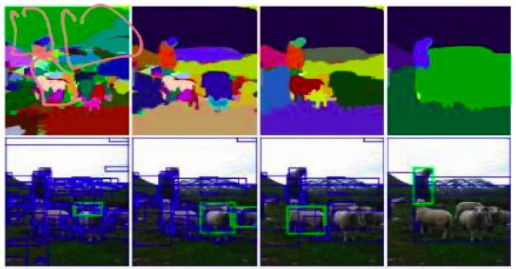
1)Finding bounding box(i.e 4 coordinates->regression problem)

2)Finding class cat/dog (i.e->classification problem)

GENERAL PIPELINE:

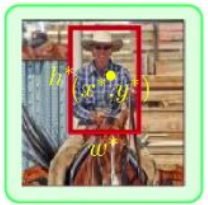
Taking example of RCNN's





Region Proposals

Proposed Box

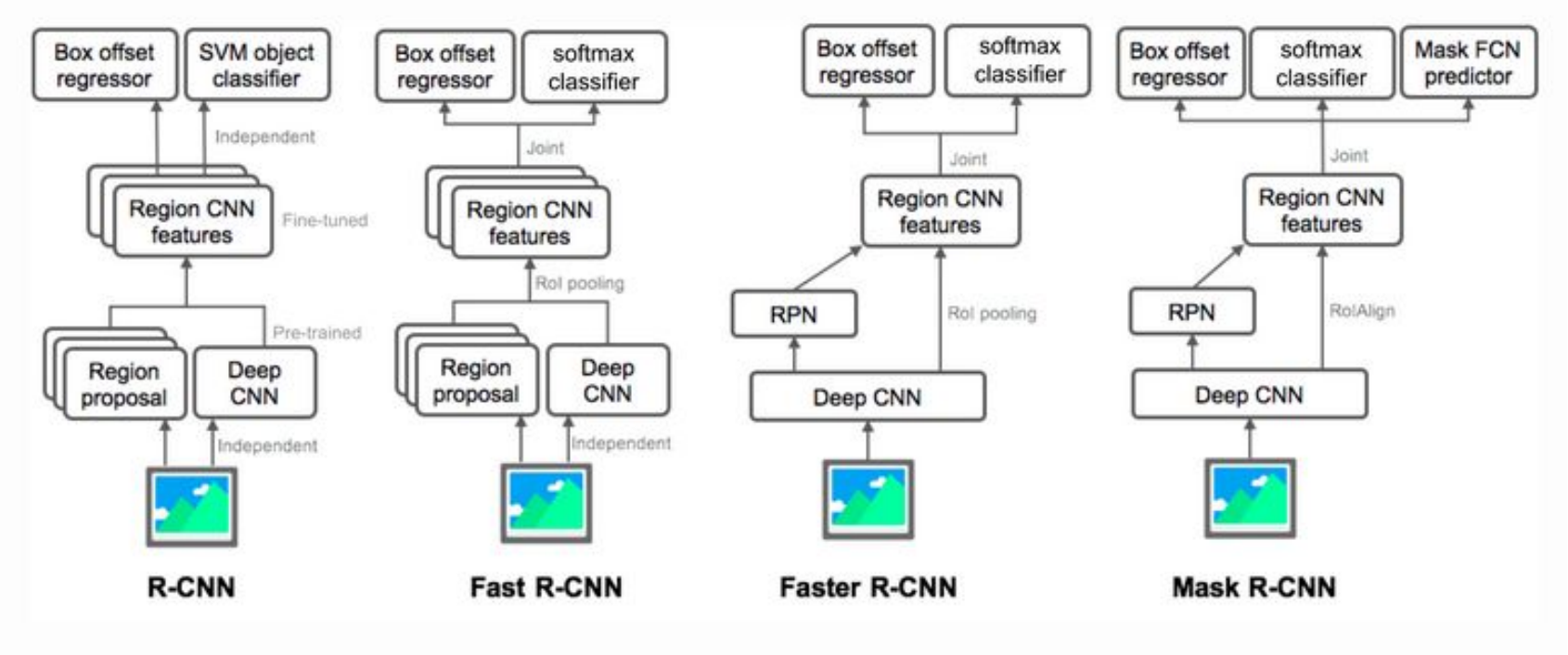


True Box

BBR

https://arxiv.org/abs/1311.2524

Various iterations after



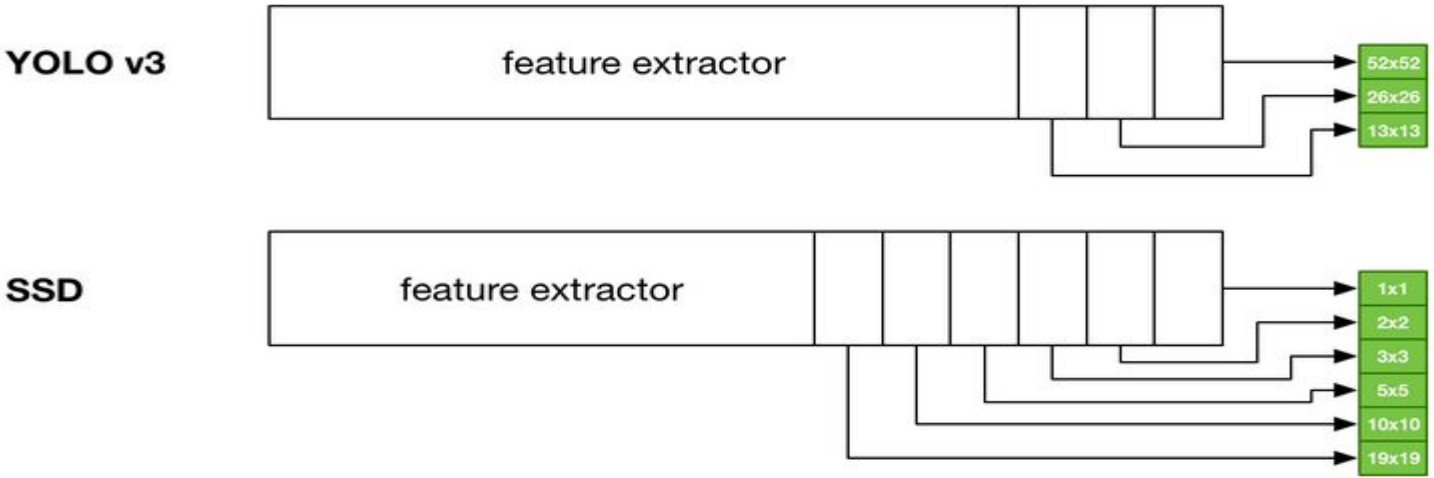
Faster-RCNN's eliminated need of region proposals, but still is computationally expensive

https://arxiv.org/abs/1506.01497

Enter Single Stage Detectors

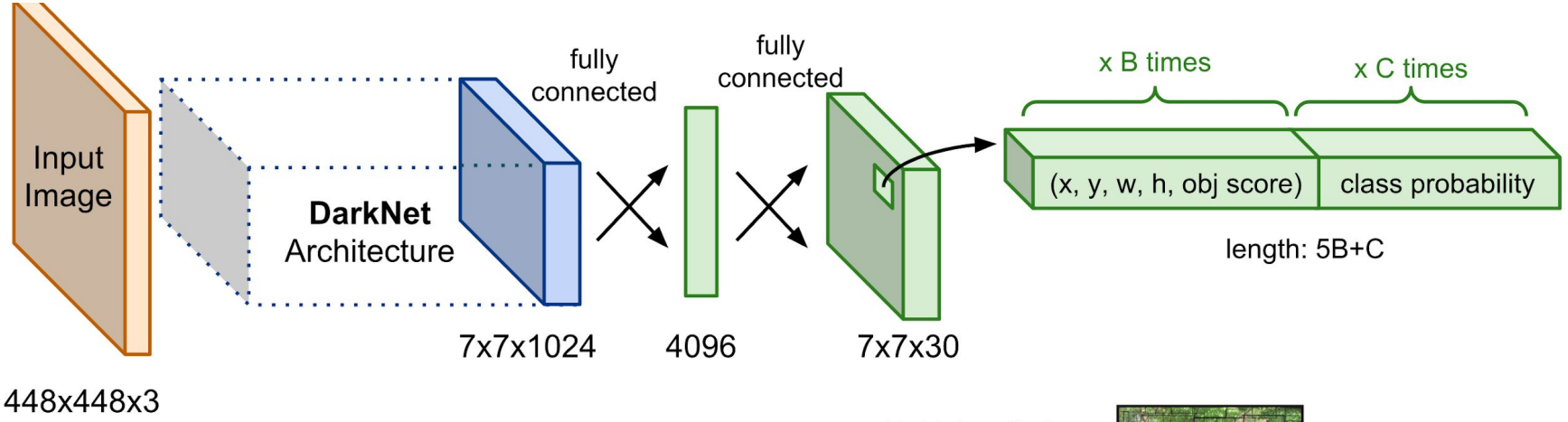
Skip the region proposal stage and runs detection directly over a dense sampling of possible locations.

Faster and simpler, but might potentially drag down the performance



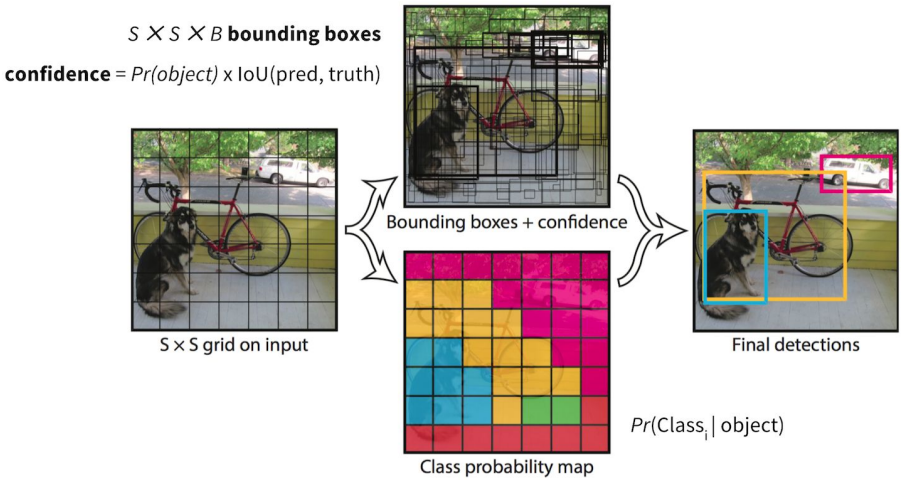
Observe: High level YOLO v3 and SSD are similar, but they arrive at their final grid sizes through different methods (YOLO uses upsampling, SSD downsampling)

YOLO



>Instead of Region Proposals, it will divide image into grids,

>determine if each grid is center of object with offset computation of ground truth boxes, to create "anchors "

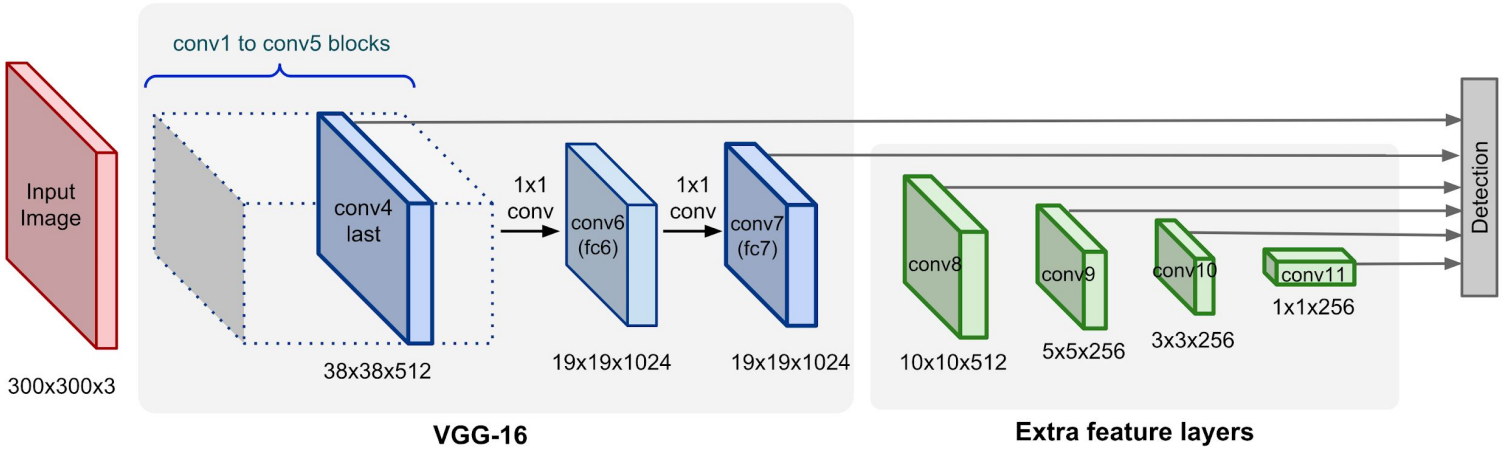


https://arxiv.org/abs/1506.02640

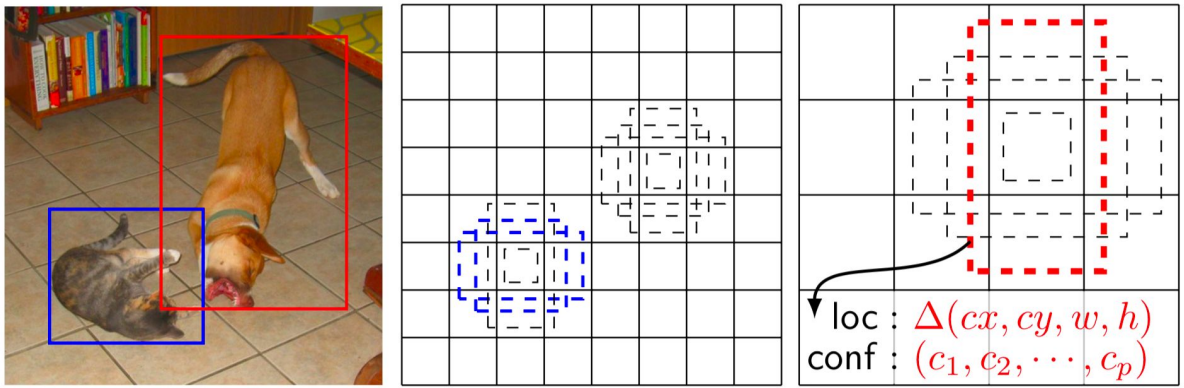
SSD’s

SSD has several grids of different sizes. The MobileNet+SSD version has 6 grids with sizes 19×19, 10×10, 5×5, 3×3, 2×2, and 1×1

SSD does not split the image into grids but predicts offset of predefined *default boxes(anchors)* for every location of the feature map.



https://arxiv.org/abs/1512.02325



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

(a) The training data contains images and ground truth boxes for every object.
(b) In a fine-grained feature maps (8 x 8), the anchor boxes of different aspect ratios correspond to smaller area of the raw input.
(c) In a coarse-grained feature map (4 x 4), the anchor boxes cover larger area of the raw input

https://arxiv.org/abs/1512.02325

Feature Extractors

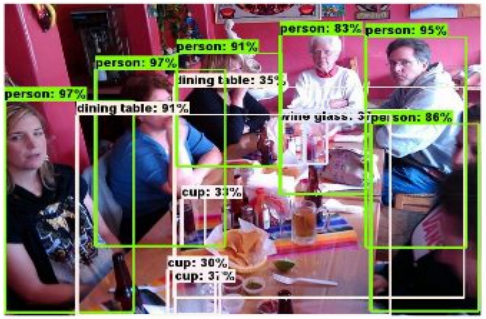
- Traditional CNN architectures, such as
  - ResNet
  - VGGNet
  - GoogleNet
  - Mobilenets etc

- Feature extractors impact both accuracy and speed. ResNet and Inception are often used if accuracy is more important than speed. MobileNet provides lightweight detector with SSD.

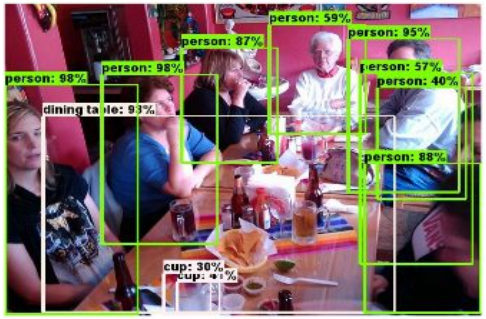
- For faster R-CNN the choice of feature extractors has significant impact on accuracy comparing with SSD.

Model	Top-1 accuracy	Num. Params.
VGG-16	71.0	14,714,688
MobileNet	71.1	3,191,072
Inception V2	73.9	10,173,112
ResNet-101	76.4	42,605,504
Inception V3	78.0	21,802,784
Inception Resnet V2	80.4	54,336,736

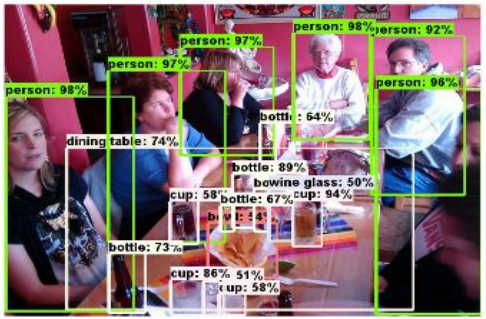
https://arxiv.org/pdf/1611.10012.pdf



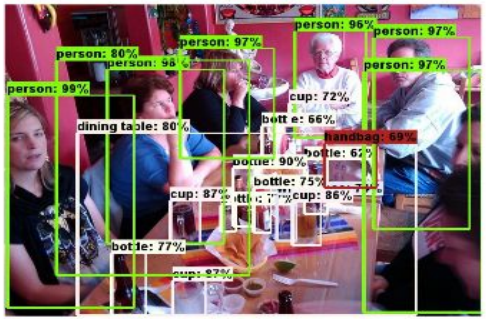
(a) SSD+Mobilenet, lowres



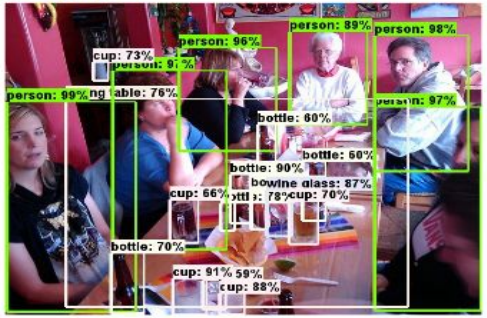
(b) SSD+InceptionV2, lowres



(c) FRCNN+Resnet101, 100 proposals



(d) RFCN+Resnet10, 300 proposals

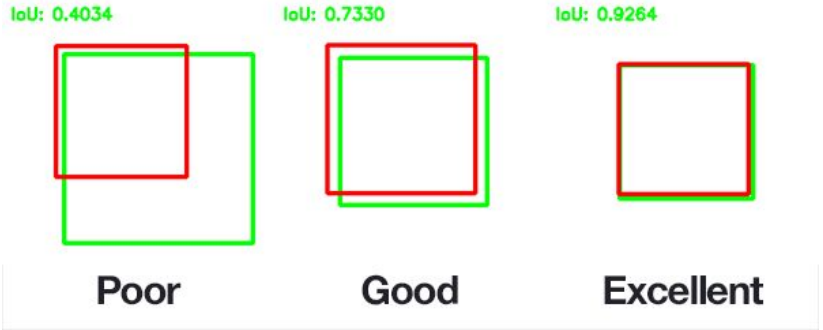


(e) FRCNN+IncResnetV2, 300 proposals

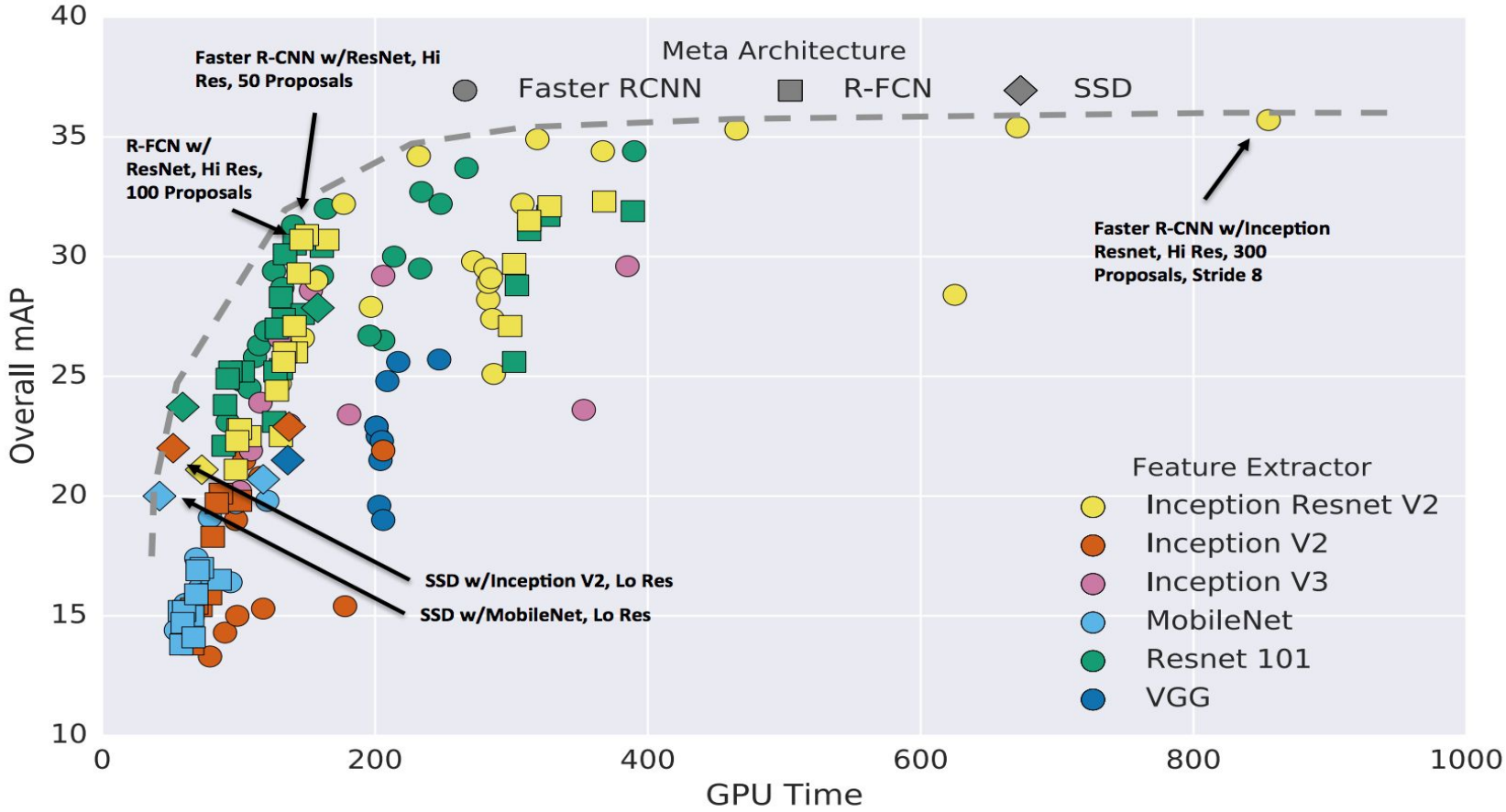
https://arxiv.org/pdf/1611.10012.pdf

Metrics

- **IoU**: A detection is a true positive if it has **“intersection over union” (IoU)** with a ground-truth box greater than some threshold



- **mAP**: Combine all detections from all test images to draw a precision-recall curve (PR curve) for each class; The “average precision” (AP) is the area under the PR curve
- **Speed**: time taken for single forward pass



In general, Faster R-CNN is more accurate while R-FCN and SSD are faster. Faster R-CNN using Inception ResNet with 300 proposals gives the highest accuracy at 1 FPS. SSD on MobileNet has the highest mAP within the fastest models

https://arxiv.org/pdf/1611.10012.pdf

Now you can understand model parameters

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

These are various pretrained models made available by tensorflow

Naming pattern:
<object-detector><base network><version><if quantised><dataset-trained upon>

Some terms :

COCO: Datset(Common Object in Context)(http://cocodataset.org/)
Quantised: Lightweight pruned models
Lowproposals : Less RP's used for faster inference(for RCNN models)

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Starting with project..

1) Defining problem statement

- Distance from camera
- Input resolution
- Object Complexity
- Background complexity
- Hardware for inference
- FPS
- Amount of data available

2)Tools

You can train your models manually, fine tuning them as per needs, but a very useful API provided by tensorflow.

https://github.com/tensorflow/models/tree/master/research/object_detection

Based on top of “TF-slim”: API defining, training and evaluating complex models, mostly used for CNN’s

Follow installation

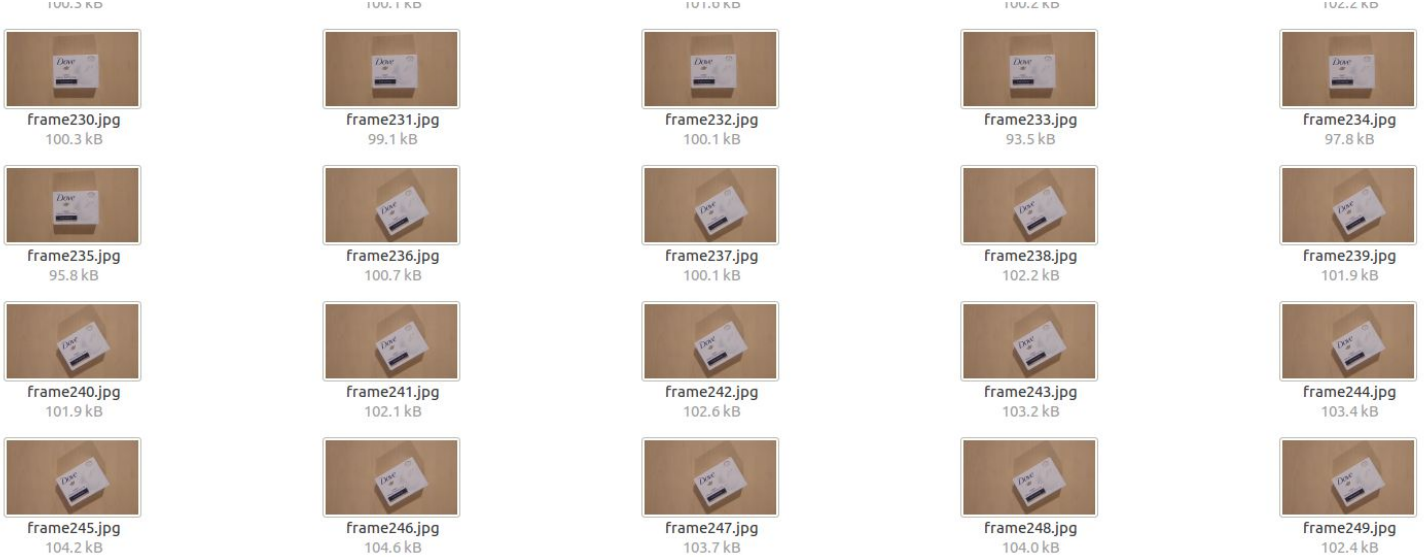
Home directory: ~/models-master/research/object-detection$

Overall Pipeline.

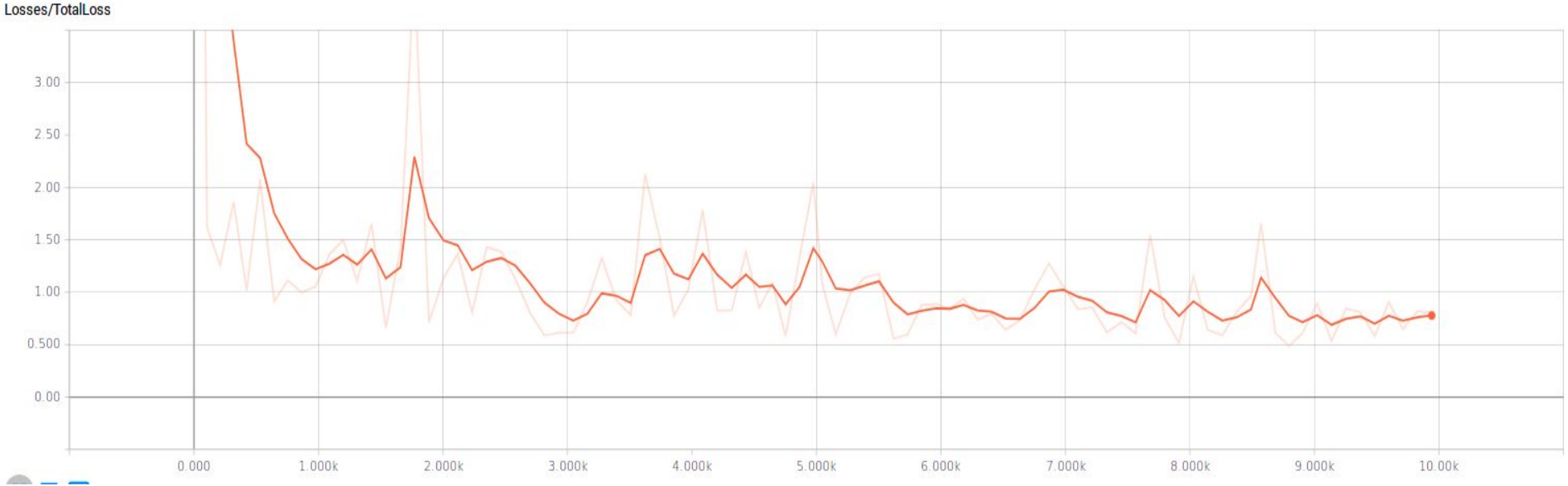
- Image Data Collection
- Resizing and scaling
- Labelling of images
- Generate TFRecord files.
- Configuring training
- Training and generating model graph
- Convert model graph to model file(.pb)
- Inferencing

3) Collecting data

Video skip frames method(if background remains constant)



But total loss curve I got is..

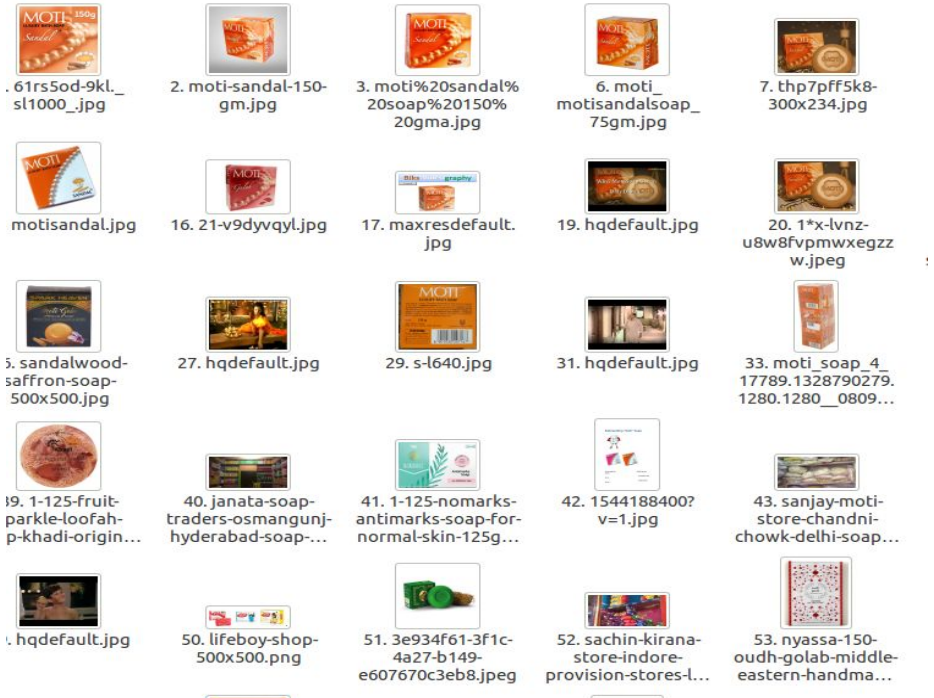


Web-scraping

https://github.com/hardikvasa/google-images-download#installation

Use proper specific keywords

Cleaning of data required



Data Augmentation

Two types:

Offline: prior augmentation- increases dataset.

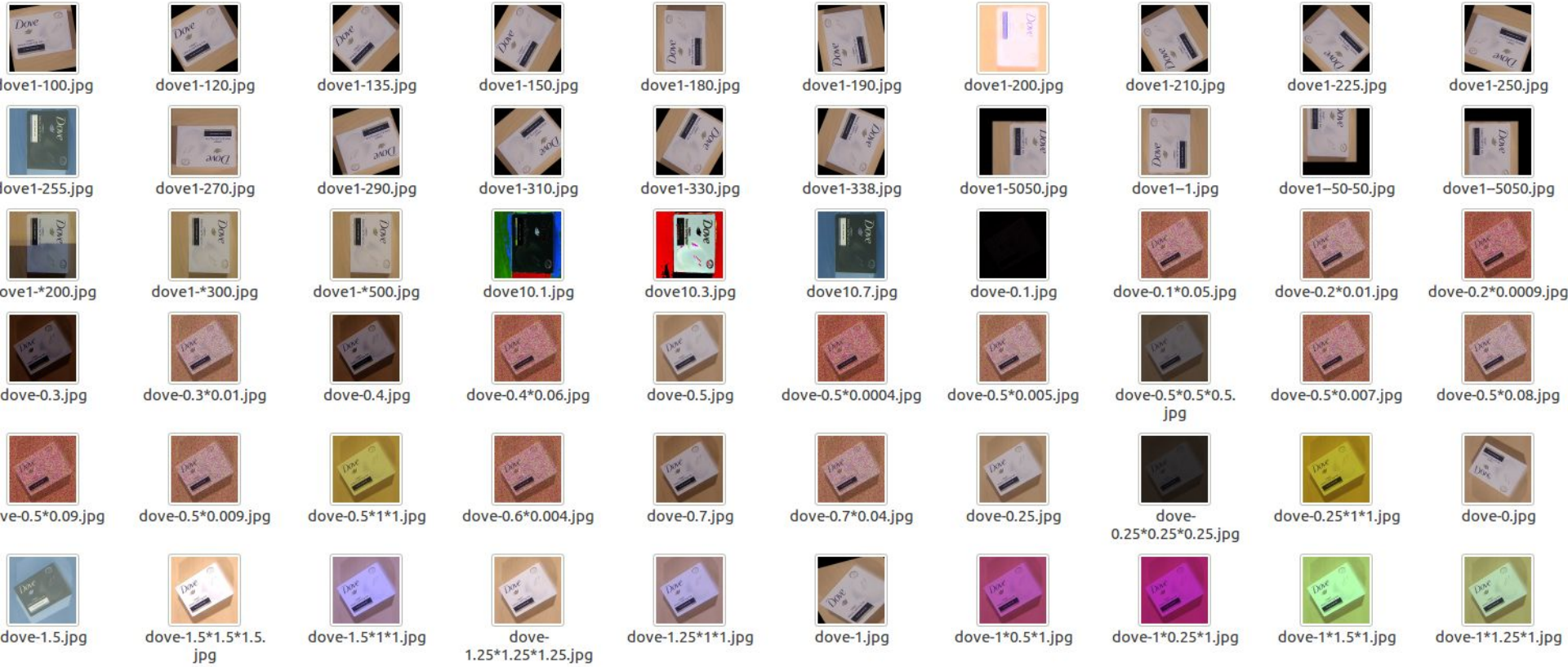
Most common are noise, scaling, flipping, cropping, brightness, contrast etc.

>script:pre-~/pre-processing/augementation.py

Online/inplace: doing with mini-batch, at time of feeding to network.

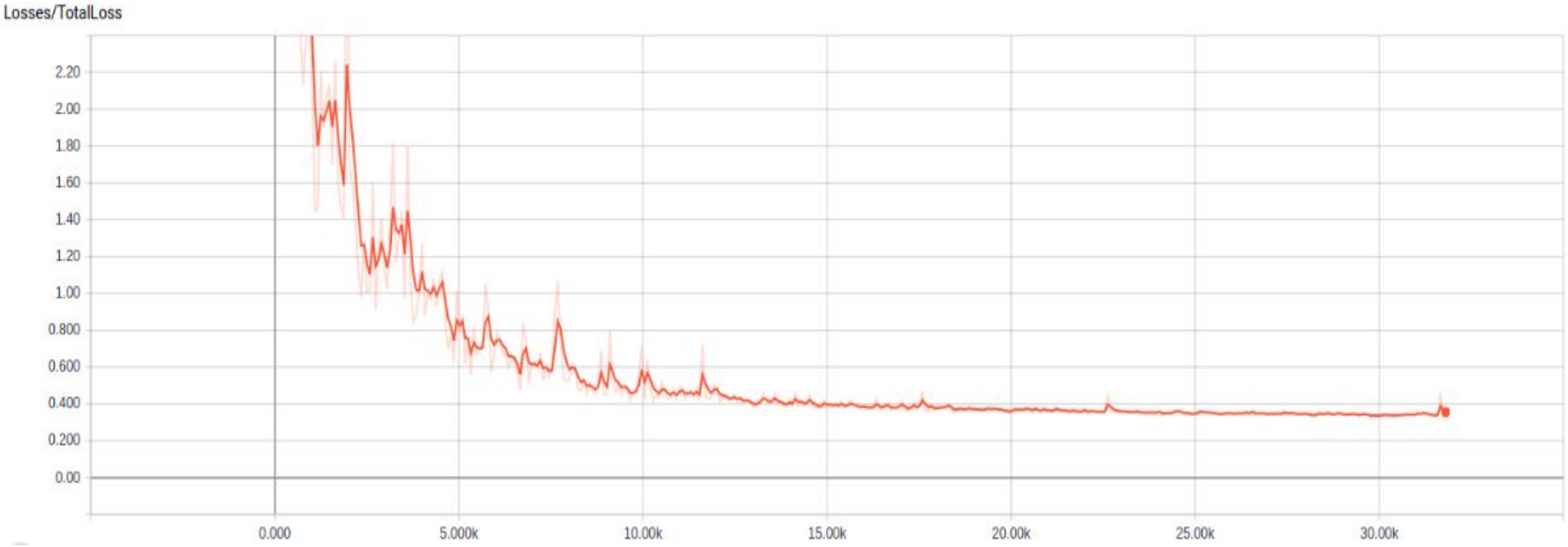
Will see in configuring training section

Augmented Samples



Used: Random brightness,hue,contrast, gaussian blurring , salt-pepper noises etc.

Result with augmentation



Still not very promising, realised the data is not inclusive enough of variations

Manually:

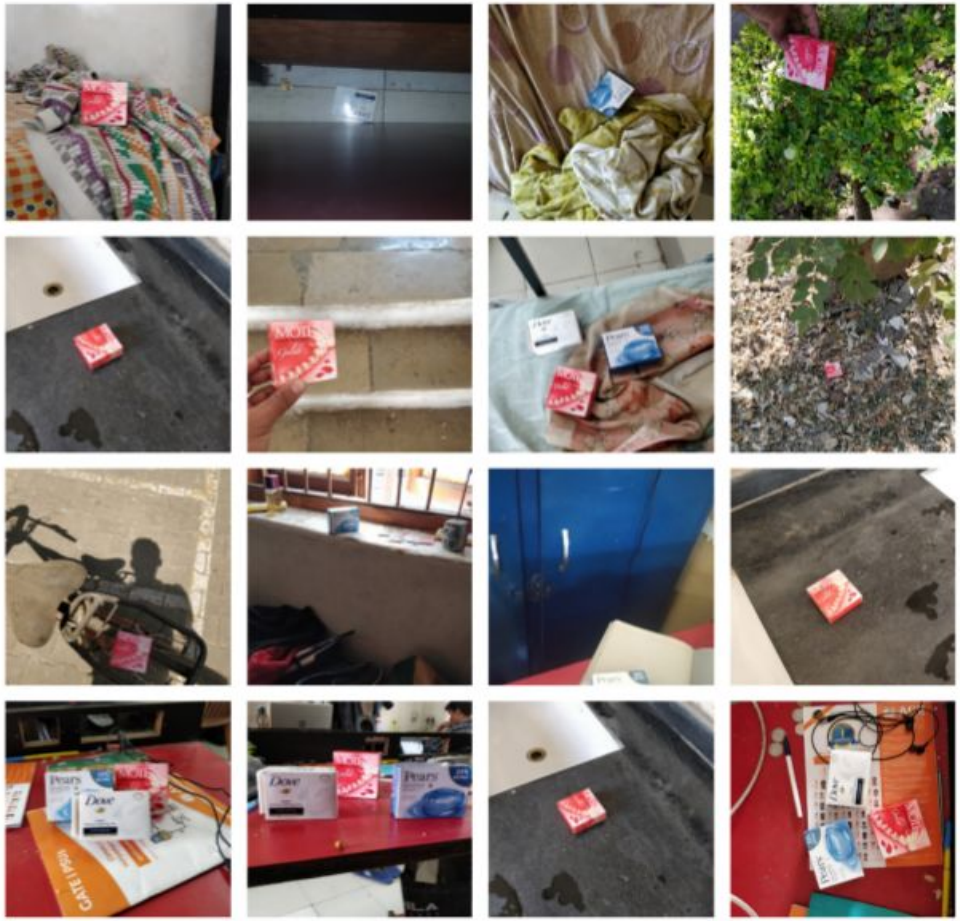
Want to emulate real life inference scenarios

Try to get more than once class in single frame

As much background variation as possible

Avoid class imbalance

Recommended at least 200 instance per class for SSD models



Data processing

- Renaming! Avoids confusion
  - ~/pre-processing/rename.py
- Resizing
  - Try to get sizes in 256-600 pixels
  - Try to conserve aspect ratio
  - Below <200kb
- Split 20% into test , 80% into train.
  - Maintain this ratio **per class** too
  - Maintain this data in an excel sheet

Labelling

https://github.com/tzutalin/labelImg

Try to cover maximum area possible in rectangular region

Particularly include edges

Use shortcuts to make it fast(w,a,s,d)

Make sure the NAME is similar to image while saving

Generate XML with file name and its bounding boxes


soap1.jpg

soap1.xml


soap2.jpg

soap2.xml


soap3.jpg

soap3.xml


soap6.jpg

soap6.xml


soap7.jpg

soap7.xml


soap8.jpg

soap8.xml


soap11.jpg

soap11.xml


soap12.jpg

soap12.xml


soap13.jpg

soap13.xml


soap16.jpg

soap16.xml


soap17.jpg

soap17.xml


soap18.jpg

soap18.xml


soap21.jpg

soap21.xml


soap22.jpg

soap22.xml


soap23.jpg

soap23.xml


soap26.jpg

soap26.xml


soap27.jpg

soap27.xml


soap28.jpg

soap28.xml

- Generate CSV files from XML for easy parsing
  - ~/pre-processing/xml_to_csv.py
  - Extract count of each class from this csv file for reference

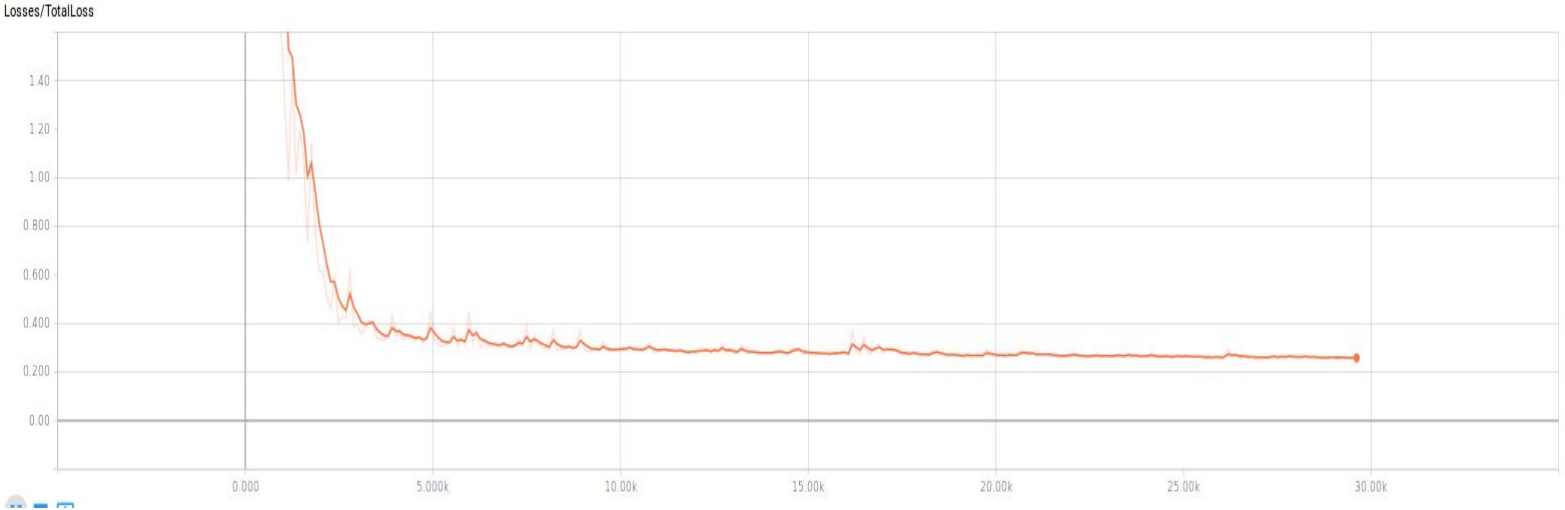
- Generate TFrecord files
  - API takes input in this specific format only
  - These .record files consists of binarised image, with all the information included
  - ~/pre-processing/generate_tfrecord.py

- Create label map

4)Training

- Download config file of chosen model
  - https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs
  - Assign proper paths for TF record files
  - Set num_classes
  - Load previous checkpoint for transfer learning
  - Set batch size
  - Start by giving training directory and config file
    - python object_detection/legacy/train.py --logtostderr --train_dir=object_detection/training/ --pipeline_config_path=object_detection/training/ssdlite.config
  - Observe using tensorboard:
    - tensorboard --logdir='<directory name>'
  - For mobilenet-ssd, starts at 15-20, train till loss hits 1-2
  - For RCNN modes, starts at 3-5, train till 0-0.5

Final training




Pears
33% EXTRA
medimix


MOTH
Pears
33% EXTRA

5)Post training

- Make sure to stop training when NOT in middle of storing checkpoint
- Make note of latest checkpoint file
- Every tensorflow model is stored as a graph. Called inference graph.
- Convert to model file
  - python export_inference_graph.py --input_type image_tensor --pipeline_config_path training/ssdlite.config --trained_checkpoint_prefix training/model.ckpt-29540 --output_directory allfour_inference_graph
- You would get a .pb model file
- Name training folders appropriately (<data><changes<number of steps>),easier when retraining

Inferencing

- Give label path, model paths in
  - ~/Object-detection-webcam.py
- Inferencing tricks

  >OpenCV scripts

  >Fastest inference use Opencv's DNN module(https://github.com/opencv/opencv/tree/master/samples/dnn)
- Can use model compression techniques
  - MorphNet(Google's architecture independent)
  - OpenVino(Intel hardware)
  - ArmNN(ARM chips)
  - TensorRT(Nvidia chip gpu's)

Lessons learned

- Complex feature extractors like ResNet and Inception-ResNet are highly accuracy if speed is not a concern.
- Experiment different feature extractors to find a good balance between speed and accuracy. Some light weight extractors make significant speed improvement with tolerable accuracy drop
- At the cost of speed, higher resolution input images improves accuracy, in particular for small objects
- Fewer proposals for Faster R-CNN can improve speed without too much accuracy drop.(https://arxiv.org/pdf/1611.10012.pdf)
- Single shot detectors tend to have problems for objects that are too close or too small.
- For large objects, SSD performs pretty well even with a simpler extractor. SSD can even match other detector accuracies with better extractor. But SSD performs much worse on **small objects** comparing to other methods.
- Feature extractor is significant speed bottleneck.

Thank you!