CS4782 Final Project:

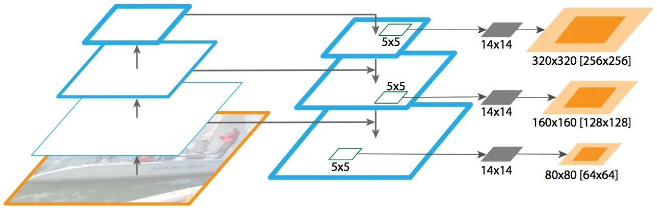
Feature Pyramid Networks for Object Detection (2016-17)

Originally authored by Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie

As reimplemented by Jack Jansons (jcj59), Rob Mosher (ram487), Raphael Thesmar (rft38)

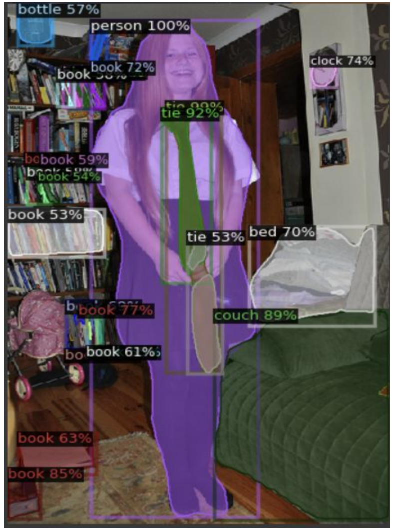
Overview/Abstract:

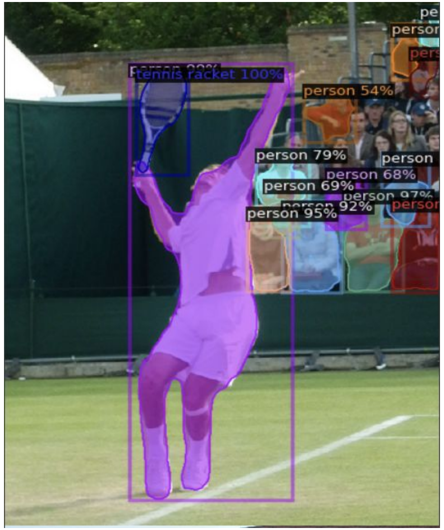
Problem Introduction:



- Recognizing objects at vastly different scales is a fundamental challenge in computer vision
  - E.g. Self-driving cars must be able to detect objects from both up close and afar in order to perform well
- Leveraging feature pyramids built upon ConvNet-based image pyramids can be a solution to this
  - an object's scale change is offset by shifting its level in the pyramid
  - Smaller objects are captured by the early layers, larger ones by the later layers
- The create an architecture (FPN) that combines low-resolution, semantically strong features with high-resolution, semantically weak features via a top-down pathway and lateral skip connections.

Object Detection using FPN and Fast R-CNN



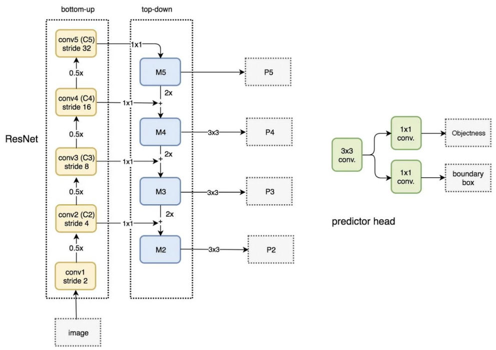


Architecture Overview

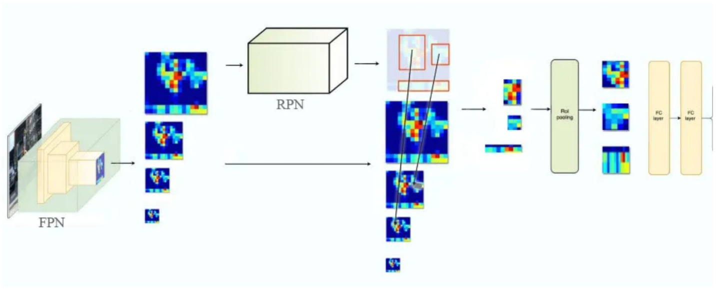
Architecture Overview:

- There are 3 main stages in the pipeline from initial image to object detection/classification:
    - Feature Pyramid Network -> Feature Maps
    - Region Proposal Network -> Proposed ROIs (Objectness + Bounding Box)
    - Faster R-CNN Detector -> Object Classifications

FPN:

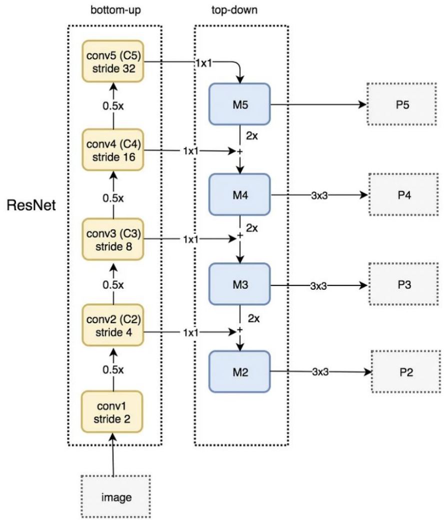


Whole System:

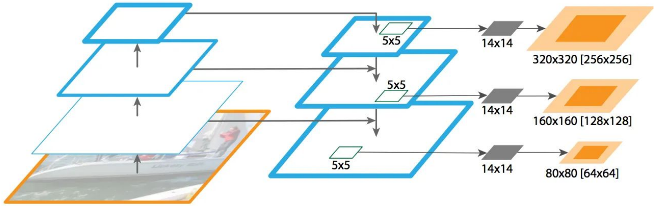


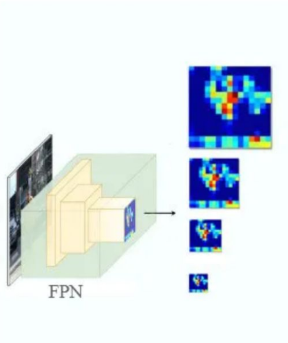
1. Feature Pyramid Networks

- The FPN is the first stage of the architecture, taking in input images and outputting feature maps on a variety of scales
- 2 stages to the FPN: Bottom-Up and Top-Down Pathways
- Each level 0.5x/2x the previous one in order to capture semantic information on a variety of scales within the original image



FPNs (Cont.):





Bottom-Up Pathway:

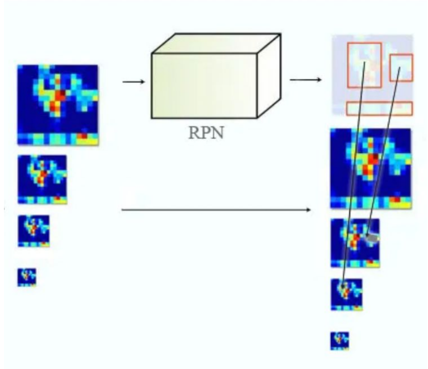
- Multiple custom blocks, each 0.5x scaled from the previous block
- Each block is composed of numerous layers with the same footprint, other than the scaling layer
    - The paper found that simple pre-trained ResNet implementations worked best
    - The scaling layer has a stride of (2,2) relative to the output of the previous block
- The output of the last layer is input into both the next block and the skip connection

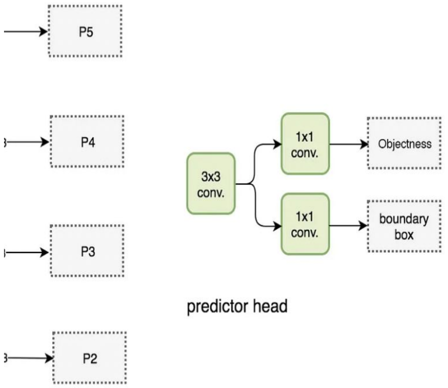
Top-Down Pathway:

- Multiple single-layer levels
- The last bottom-up block output is passed through a 1x1 convolution to standardize dimensionality
- Each layer consists of an element wise addition of the 2x nearest-neighbor upscale of the previous layer and the corresponding skip connection

2. Region Proposal Networks

- A sliding window predictor head then passes over the FPN feature maps
- This predictor head consists of a 3x3 convolutional layer and two twin 1x1 convolutional layers
- The twin 1x1 conv. layers output predictions on objectness and object boundary box at each location, respectively
- The most likely ROIs are tracked and fed into Faster R-CNN for classification

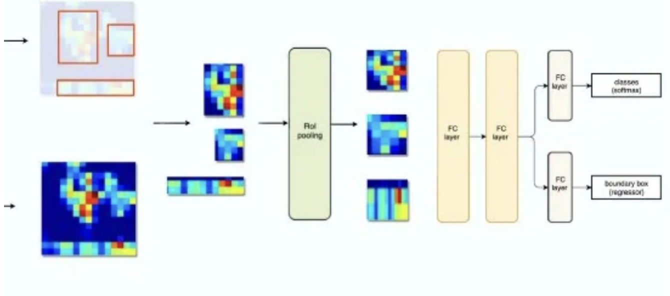




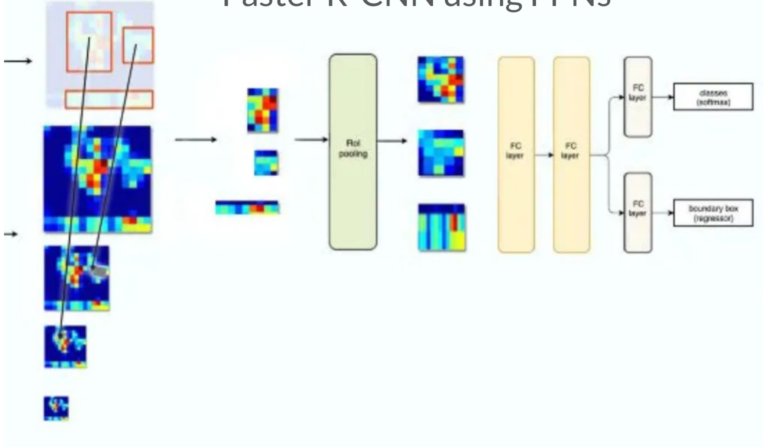
3. Faster R-CNN with FPNs

- Faster R-CNN is an industry standard object detection algorithm
  - Standard F.R-CNN usually relies on standard ConvNets for generating feature maps used for determining ROIs
  - This only captures information at one scale and can lead to performance issues when objects of different scale exist in one image
- Faster R-CNN w/FPNs seeks to fix this!
  - ROIs are assigned to feature pyramid levels depending on their width/height

Standard Faster R-CNN



Faster R-CNN using FPNs



Implementation Details

Bottom Up Implementation

Custom Backbone

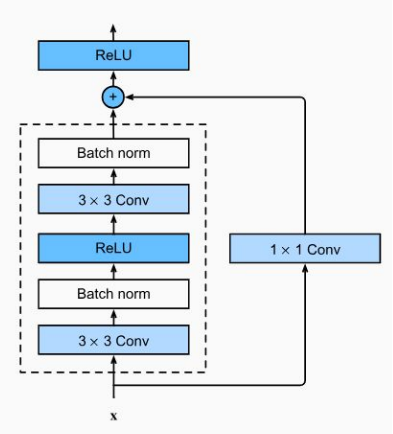
- Implemented custom “bottom up” block
    - See image to right for architecture
    - Each block contains one convolution with stride 2 to reduce dimensions by 2
    - Each block outputs 3 channels

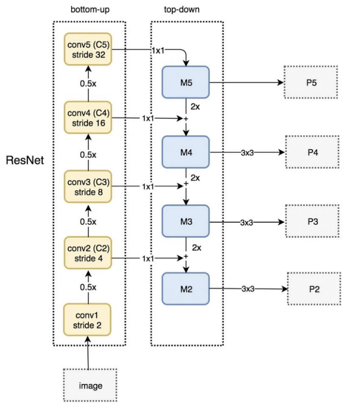
ResNet50

- Use pretrained weights
- 5 Layers
    - Layer 1: Simple 3x3 conv with 64 output channels
    - Layer 2: 3 residual blocks with 256 output channels
    - Layer 3: 4 residual blocks with 512 output channels
    - Layer 4: 6 residual blocks with 1024 output channels
    - Layer 5: 3 residual blocks with 2048 output channels
- Memory Issues! Model too complex

ResNet34

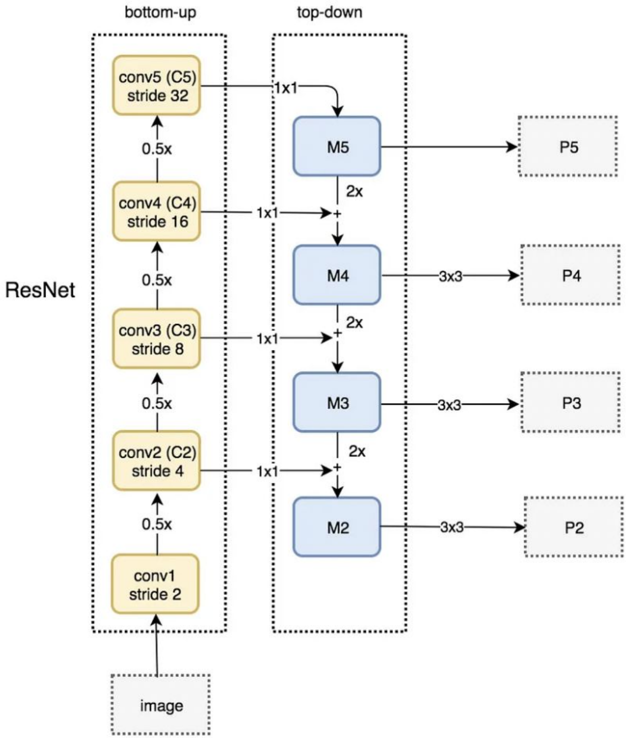
- Same architecture as ResNet50
- Output channels 64, 64, 128, 256, 512 respectively
- Resolves memory issue!





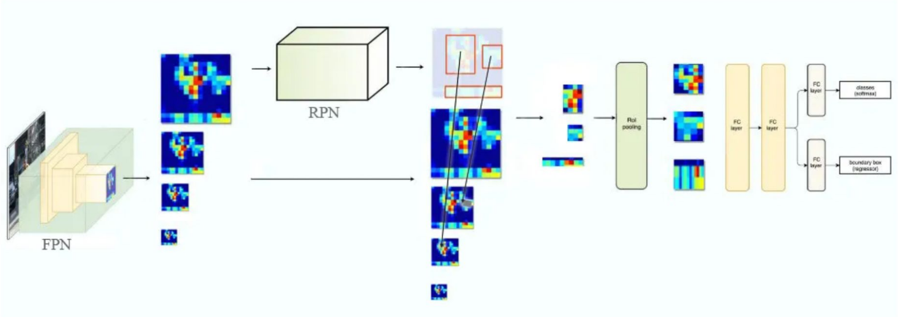
Top Down Implementation

- 1x1 convolutions for residual connections
  - Input channels match bottom up layers
  - Outputs 256 channels
- PyTorch's Upsample function
  - Nearest neighbor upsampling
  - Double dimensions to match bottom up layer
  - Add cropping function for odd dimensional layers
  - Maintains channels
- 3x3 convolutions for final outputs
  - Takes in 256 channels
  - Outputs 256 channels



RPN and Faster R-CNN using Detectron2

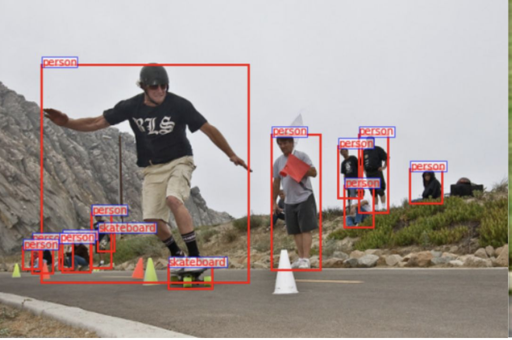
- Use Facebook AI Research's Detectron2 package built for object detection and other vision tasks
- Feed our bottom-up and top down architecture into prebuilt RPN and Faster R-CNN architectures
- Use Detectron2 to train
    - Modify learning rate, learning schedule, batch size, iterations, image sizes
    - Automatically loads and partitions dataset
    - Built in accuracy and loss calculations
    - Automatically saves weights and metrics after each epoch



Result Replication

Dataset and Preparation

- Our experiments were performed on the 80 category COCO 2017 detection dataset
- COCO 2017 is a large-scale dataset comprised of 118k training images, 5k validation images and 40k test images
- The datasets contains classification data with bounding boxes, keypoints data, segmentation data and dense estimation of human pose data.





Training + Testing

Our ResNet implementation:

- We trained using a Colab Deep Learning VM Environment with a single T4 GPU for 36 Hours
- That's 270k total iterations, where each iteration is 5 batches of 4 images

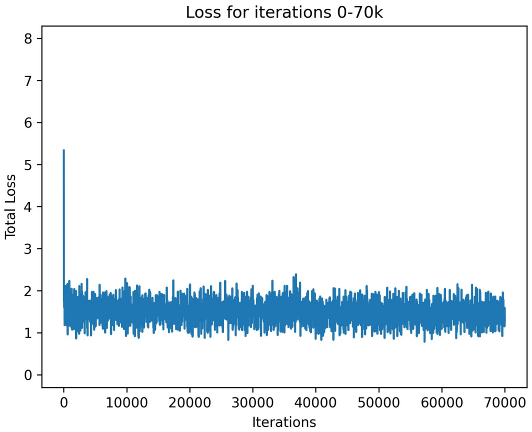
ResNet34 Implementation:

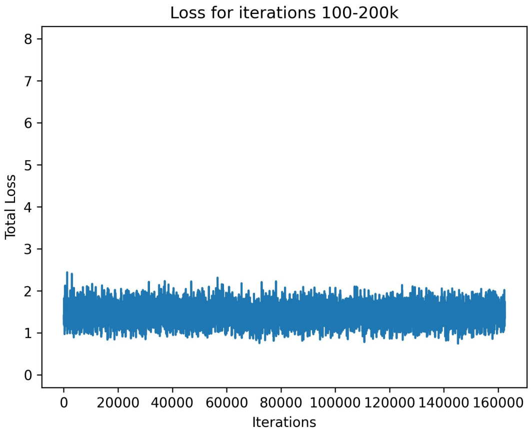
- As of last night, we were at 20k total iterations with the same settings. Now at 70k.

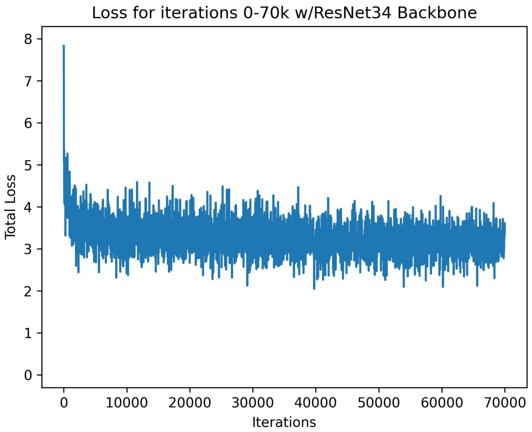
In both cases, we started with a learning rate of 2e-3 which slowly converged to 1e-4.

Loss Results

Total loss at each iteration for both of our implementations







Duplicated Results

- The goal was to replicate section 5.2: Object Detection with Fast/Faster R-CNN FPN

Note:

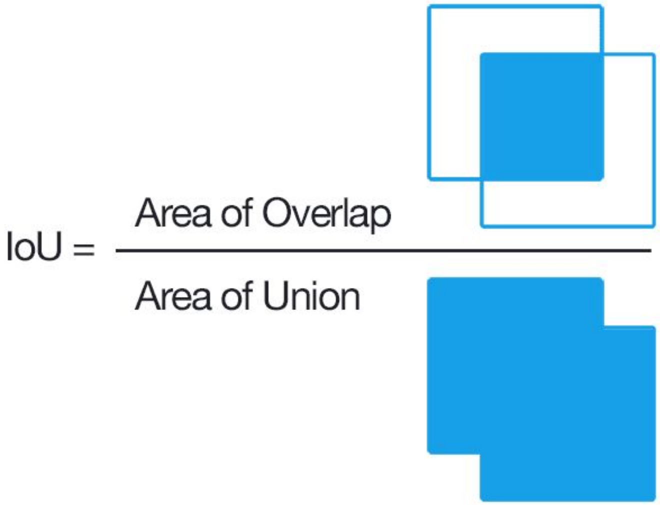
- The authors of the paper trained for 8 hours on 8 NVIDIA Tesla P100 GPUs
- We had far less compute power than they did, but we did the best we could with our options

Table 2. Object detection results using **Fast R-CNN** [11] on *a fixed set of proposals* (RPN, $\{P_k\}$, Table 1(c)), evaluated on the COCO `minival` set. Models are trained on the `trainval35k` set. All results are based on ResNet-50 and share the same hyper-parameters.

Faster R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(*) baseline from He *et al.* [16]†	RPN, C_4	C_4	conv5			47.3	26.3	-	-	-
(a) baseline on conv4	RPN, C_4	C_4	conv5			53.1	31.6	13.2	35.6	**47.1**
(b) baseline on conv5	RPN, C_5	C_5	2*fc*			51.7	28.0	9.6	31.9	43.1
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	45.8

Quick Aside

- IoU loss is Intersection over Union loss:
    - IoU Loss = (Area of Intersection)/(Area of Union)
- In particular used in image classification when dealing with bounding boxes like we are.
- Used for a lot of Computer Vision metrics with COCO like APs or ARs seen in previous slides



Explaining the Metrics

- AP stands for average precision(taken over the classes) which was traditionally called mean Average Precision. Precision is calculated as the ratio of true positives to the sum of true positives and false positives.
    - True positive occurs when the IoU with any ground truth bounding box exceeds the specified threshold
    - A false positive occurs when the IoU is below the threshold or if there is no matching ground truth bounding box
- AP@0.5, AP_s, AP_m and AP_l
    - @0.5 means at an IoU of 0.5, s=small=small objects smaller than 32^2, m=medium objects between 32^2 and 96^2 and l=larger objects with area > 96^2

Evaluation Results

- Our backbone implementation after 270k iterations:
    - $AP@0.5 = AP_s = AP_m = AP_l = 0$

- The ResNet34 implementation after 20k iterations:
    - $AP@0.5 = AP_s = AP_m = AP_l = 0$

A lot of work does have to be done…

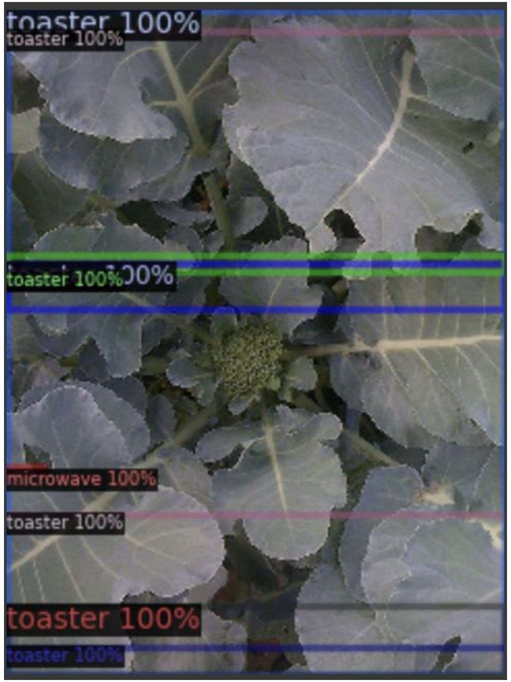
AP@0.5	AP	AP_s	AP_m	AP_l
47.3	26.3	-	-	-
53.1	31.6	13.2	35.6	**47.1**
51.7	28.0	9.6	31.9	43.1
56.9	**33.9**	**17.8**	**37.7**	45.8

Toasters + Microwaves??

- Only 2 classes represented in attempted object detection after 100k iterations
- Most things are toasters, and some things are microwaves, apparently

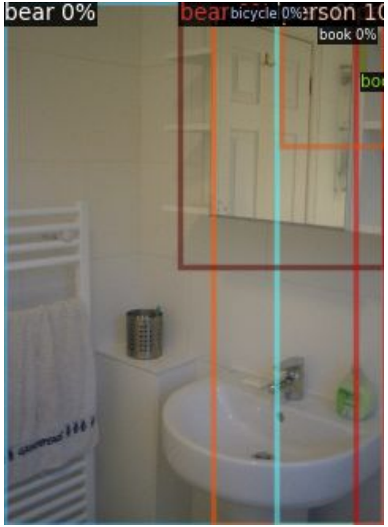
Possible reasons:

- Lack of backbone pre-training and general time/compute spent showing
- The main reason why we decided to implement the ResNet34 backbone



UnBEARable Results:

- At 200k iterations, we had more diversity in labels and bounding box shape
- Still incorrect / inconsistent results, but progress is being made!
- Similar situation with 260k iterations, with slightly more classes and better bounding boxes
- Very fond of bears this time around. It also seemed to like knives.



UnBEARable Results Again...

For the ResNet34:

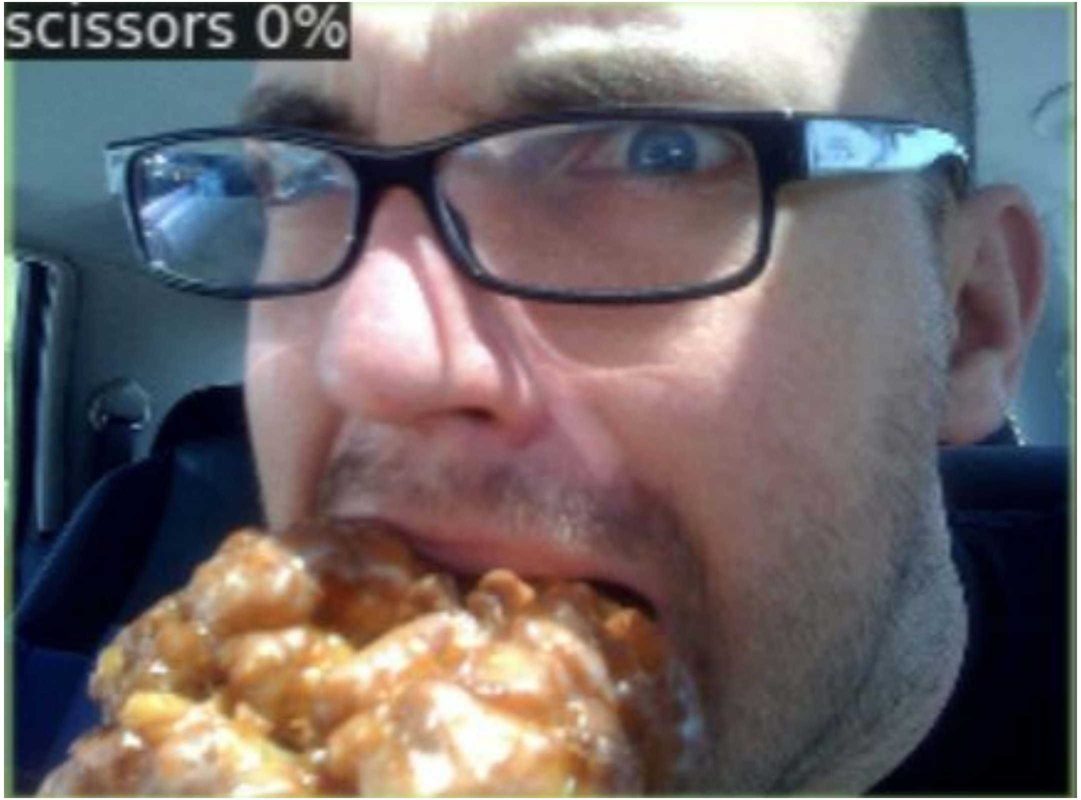
- After 20k iterations, which is much lower than the other results
- Very similar results, even less distributed classes
- After 70k, it seemed like bananas and books

Moving Forward:

- As of now, comparing our results with the table from section 5.2 is irrelevant
    - We need to be more accurate for the comparison to have any meaning
- We will attempt to overtrain our model on one batch of images to ensure our model is updating weights properly
    - Should only require 100 iterations or so to see results
    - If we still do not see relevant results, then there is an issue in our data loading or training process
- Using our experiment, we will debug our model and retrain
    - Hopefully we will see real results and improvement in the loss
    - Be able to make comparison with

Lessons Learned

- Replicating results from the large DL models can be very difficult, mostly due to the lack of compute power but also due to missing implementation details
- What architecture current state of the art computer vision algorithms, such as YOLO or Detectron2, use.
- We also learned how much hyper parameters affect performance, from the learning rate schedule to the different image size inputs allowed by the algorithm
- How the need for more efficient and faster hardware is growing in deep learning.



Questions ?