**AIRBNB Article Summary**

Data

* 43k internal images and 32k public images with 30 customized classes

Metric: mAP

Goal: mAP >= 50% (Based on Google AutoML)

Model

* Ssd\_mobilenet\_v2 = mAP 14%, learn rate = 8e-4
* faster\_rcnn\_inception\_resnet\_v2 = mAP 27%, learn rate = 6e-5
* Google AutoML= mAP 68%

Validation

* 7.5k images (10% held-out)

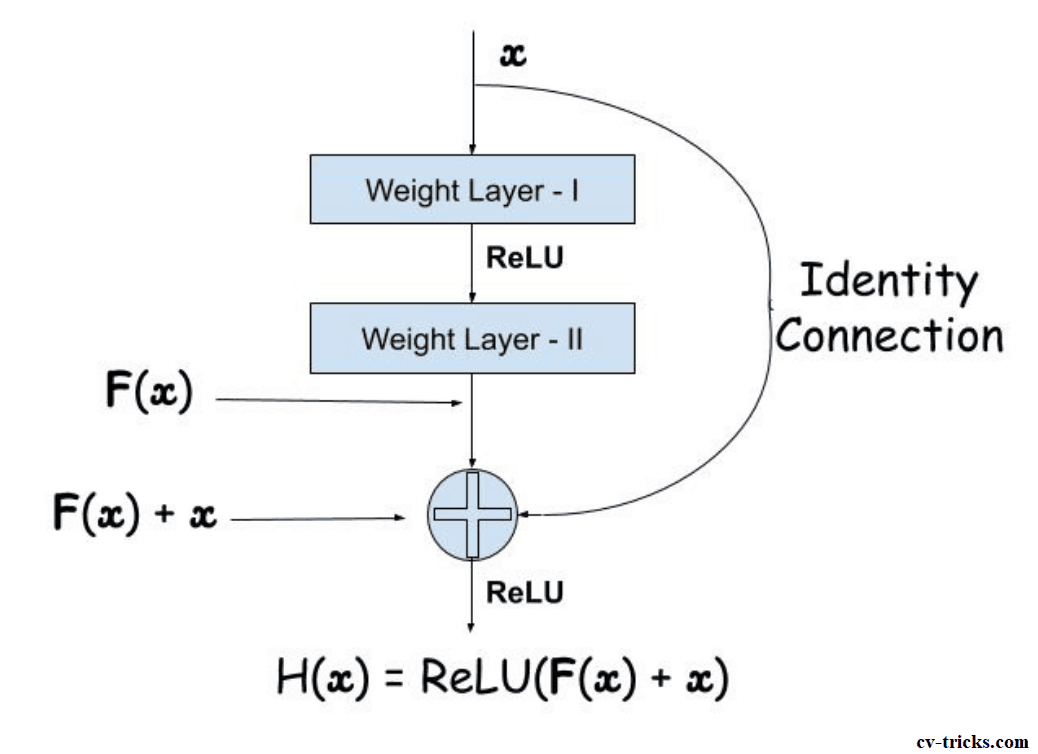
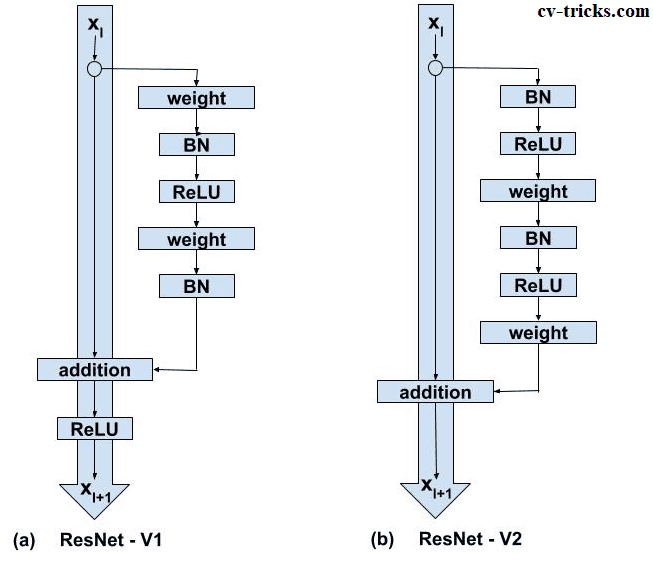
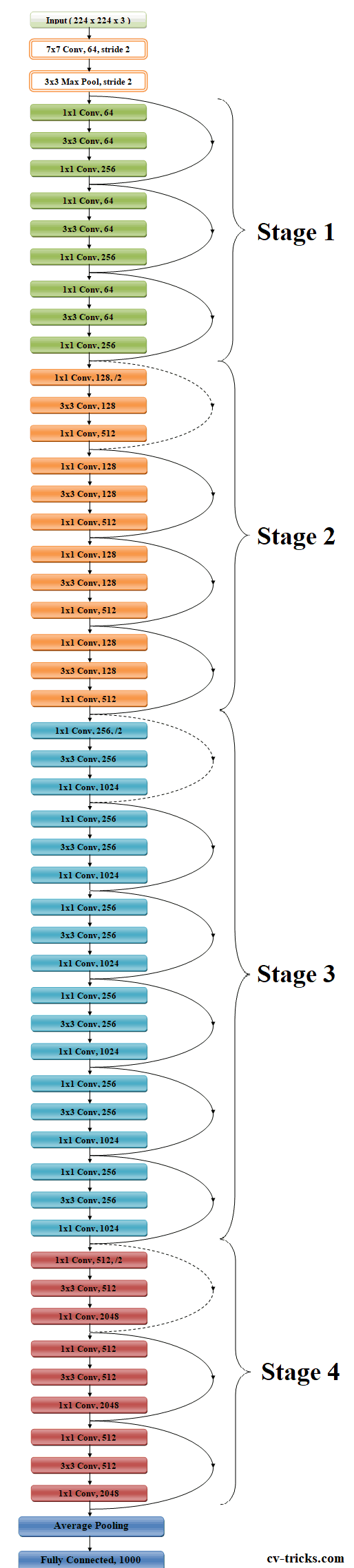
Classes (30 classes): Toilet, Swimming\_pool, Bed, Billiard\_table, Sink, fountain, oven, ceiling\_fan, Television, microwave\_oven, gas\_stove, refreigerator, Kitchen & dining room table, washing machine, bathtub, stairs, fireplace, pillow, mirror, shower, couch, countertop, coffeemaker, dishwasher, sofa\_bed, tree\_house, towel, porch, wine\_rack, jacuzzi

**mAP (Mean Average Precision)**

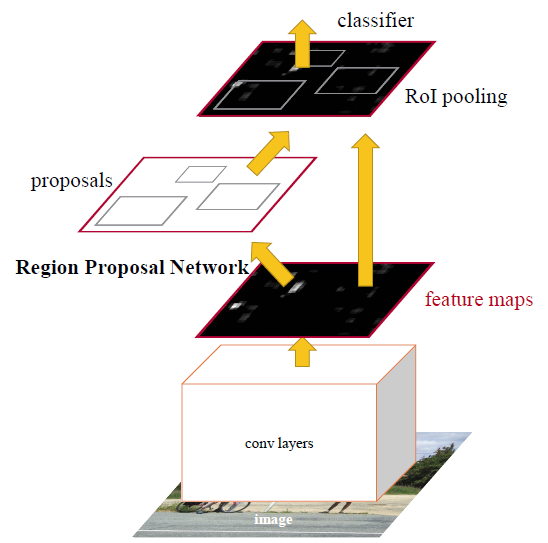
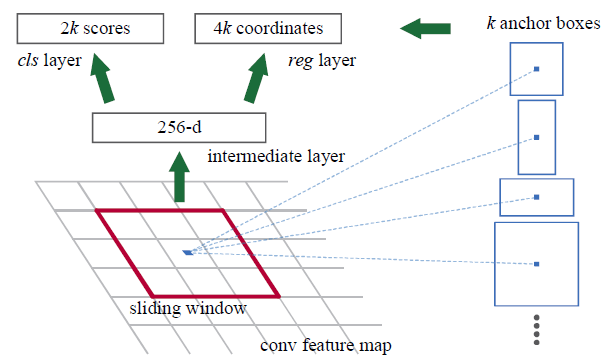
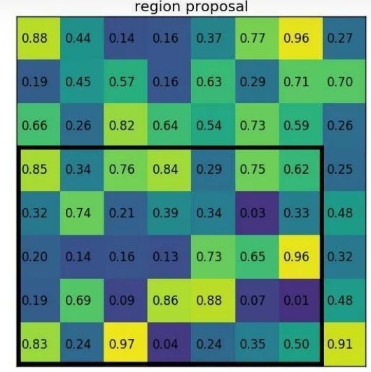
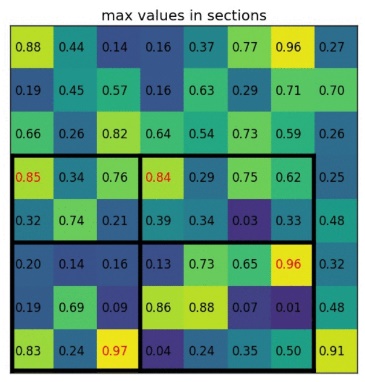
* Precision = TP / (TP + FP) -> Out of all predicted positives, how many are true positives?
* Recall = TP / (TP + FN) -> Out of all actual positives, how many positives do we obtain?
* IoU = Intersection / Union
  + IoU >= 0.5 -> True Positives (Predicted close enough to true bounding box)
  + IoU < 0.5 -> False Positives (Predicted object to be quite far from bounding box)
  + No object detected => False Negative
* COCO Evaluator:
  + AP = Avg Precision at 10 IoU Threshold from 0.05 to 0.95
  + AP50 = Avg Precision at IoU Threshold of 0.5 (Pascal VOC metric)
  + AP75 = Avg Precision at IoU Threshold of 0.75
* Mean Average Precision =

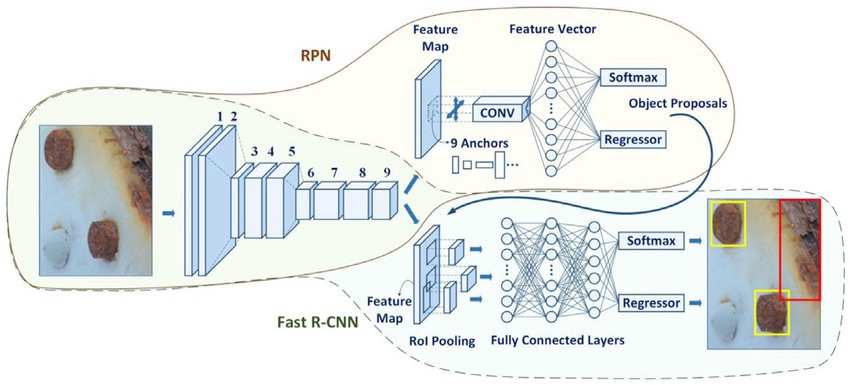
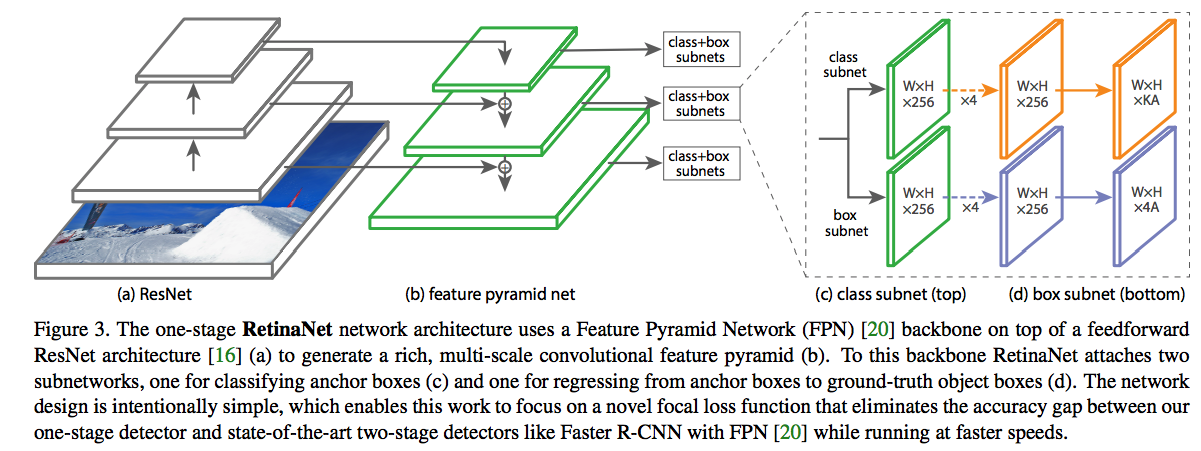
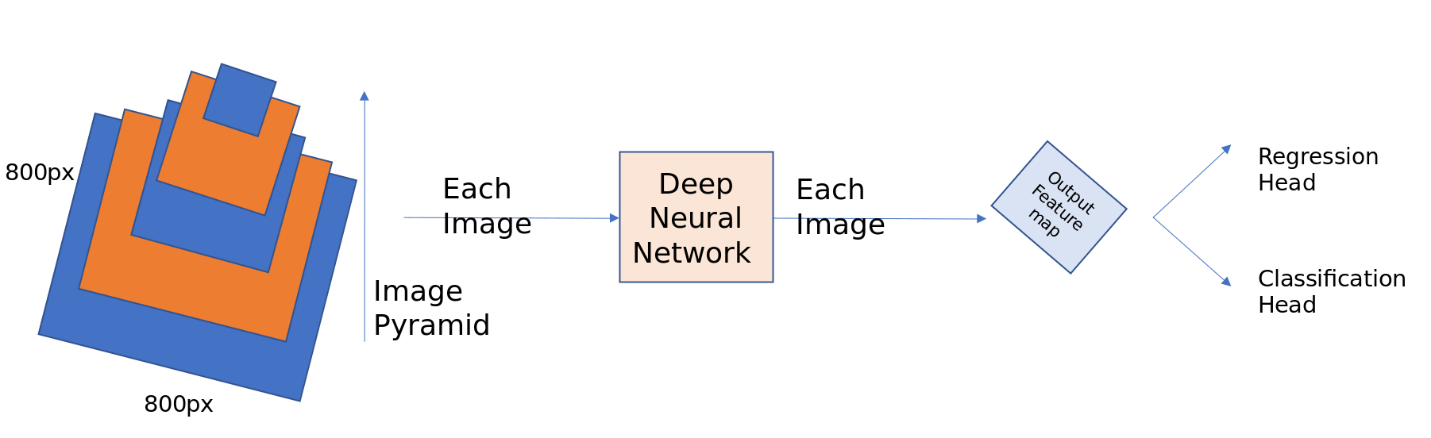
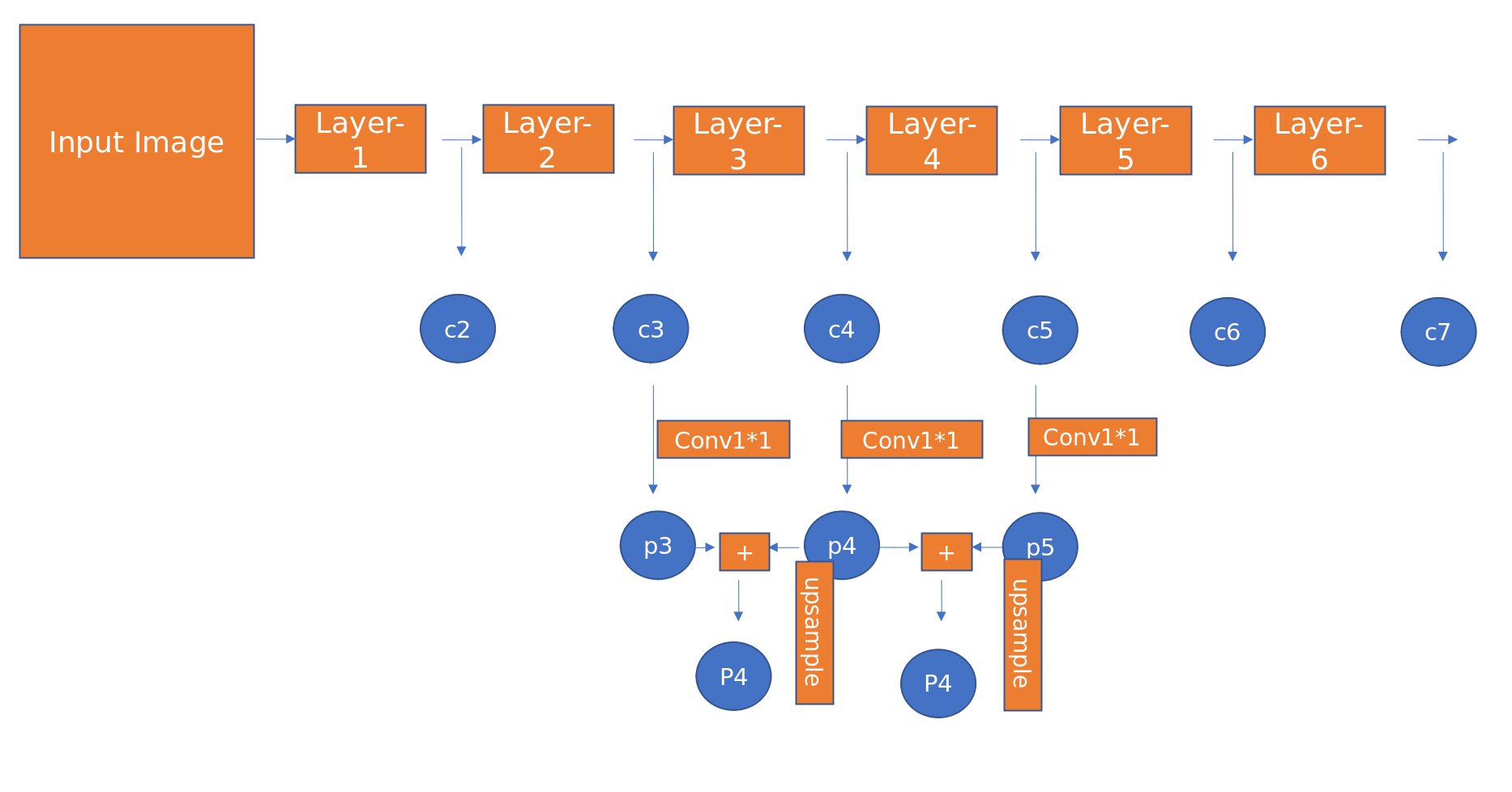
**Detectron2 Models**

* <https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md>
* **Faster R-CNN (R50-FPN, R101-FPN)**
  + Uses Resnet50 as base for R50-FPN, and Resnet101 as base for R101-FPN

In skip connections. User Resnet v2

* + In Faster RCNN, both proposal generation and object detection task are all done by the same convolution neural nets.
  + Region Proposal Network
    - K anchor boxes are used (default K = 9)
    - Outputs 2k (18) scores for cls layers, and 4k (36) coordinates for reg layers
    - RPN network is to pre-check which location contains object. And the corresponding locations and bounding boxes will pass to detection network
    - Loss RPN =
    - P\_i\* = true probability (0 or 1). 1 if there is an object 0 otherwise
    - P\_i = predicted probability of an object in box i
    - N\_cls = 2K (18 points)
    - N\_reg = 4K (36 points)
  + ROI Pooling (Region of Interest Pooling)
    - ROI pooling produces the fixed-size feature maps from non-uniform inputs by doing max-pooling on the inputs.
    - Only takes boxes from the RPN output and maps it into fixed sized output x by x
    - A picture containing cabinet

      Description automatically generatedExample: want output to be 2x2
    - Take 7x5 Region box from the RPN layer
    - Divide equally (not always even) to 2x2 smaller boxes
    - Do Max pooling and obtain output
  + After ROI pooling, connect to fully connected layers with softmax and regressor output.
  + Loss output =
  + <https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/>
* **Retinanet (R50, R101)**
  + Architecture
  + ResNet is used for deep feature extraction.
  + Only use P3, P4, P5, P6, P7 anchors of size [32, 54, 128, 256, 512] with aspect ratio [1:1, 1:2, 2:1]. So, there will be 15 anchor boxes over the pyramid at each location
  + All anchor boxes outside the image dimension are ignored
  + + if IOU of >= 0,7, - if IOU <= 0.3 with ground truth boxes (like classification for focal loss)
  + ground-truth boxes are associated with anchors, which have been assigned to pyramid levels
  + Loss function: Focal Loss function (variant of cross entropy loss)
  + Add less weights to correctly classified examples and large weights to misclassified examples.
  + Focal Loss (p\_t) = -(1-p\_t)^gamma \* alpha \* log (p\_t)
    - Gamma is focusing parameter (default = 2)
    - Alpha is balancing parameter (default = 0.25)
  + The total focal loss of an image is computed as the sum of the focal loss over all ~100k anchors, normalized by the number of anchors assigned to a ground-truth box.
  + <https://medium.com/@14prakash/the-intuition-behind-retinanet-eb636755607d>

**Project Notes**

**Training Data:** 35k images downloaded from the same 30 classes as Airbnb, 0.7k validation images, and 2k images for test and evaluation

**Models Used (Detectron2):**

* <https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md>
* Faster R-CNN: (3x learning rate)
  + R50-FPN- 3x, R-101-FPN-3x
* Retina Net (3x learning rate)
  + RN-50-3x, RN-101-3x

**Tools Used**

* OIDv4\_Toolkit: To download only several classes from open images
  + <https://github.com/EscVM/OIDv4_ToolKit>
* Detectron2: For Deep learning model (implemented in pytorch)
  + <https://github.com/facebookresearch/detectron2>
* Streamlit: Build front end machine learning application
  + <https://www.streamlit.io/>
* Google Colab: For Machine learning training
* Weights and Biases: To keep track of metrics during training
  + <https://www.wandb.com/>
* Docker: To containerize application
* Google Cloud Platform: For deployment to be accessible for public

**Procedure (Rough):**

* Download open images pictures into local file
* Convert labels to detectron2 style labels
  + Example:

{'file\_name': 'C:\\Users\\Jonathan Santoso\\workspace3\\Personal Projects\\Amenity detection\\dataset\\val\\007f71665b0812a7.jpg',

'height': 768,

'width': 1024,

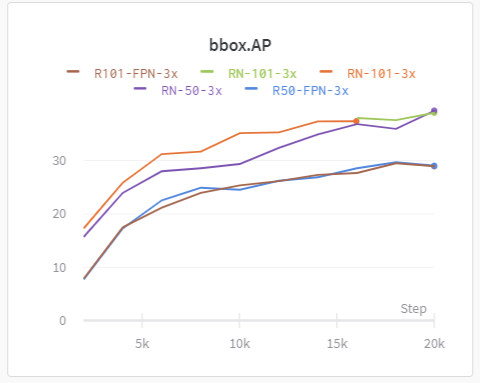
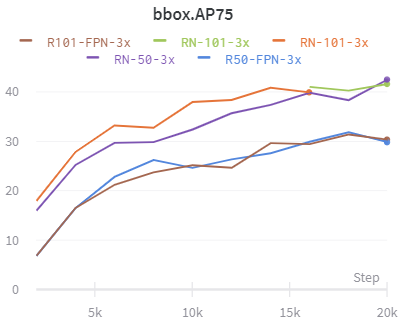
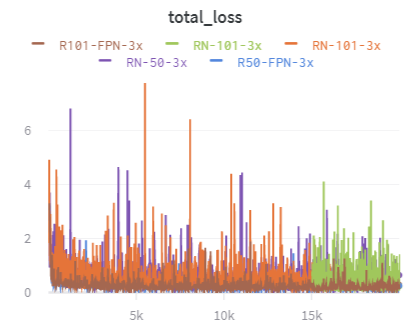
'image\_id': '007f71665b0812a7',

'annotations': [{'bbox': [0.0, 0.5952, 1024.0, 768.0],

'bbox\_mode': 0,

'category\_id': 24}]}

* Start training Detectron2 models from the given images. I trained the model for 20k iterations and evaluated performance every 2k iterations
* Obtain testing results and pick best model
* Create Front-end streamlit application
* Use Docker to containerize application
* Deploy to Google Cloud Platform

**Training Results (on Validation Set during training ~700 images):**

Training utilizes Stochastic Gradient Descent.

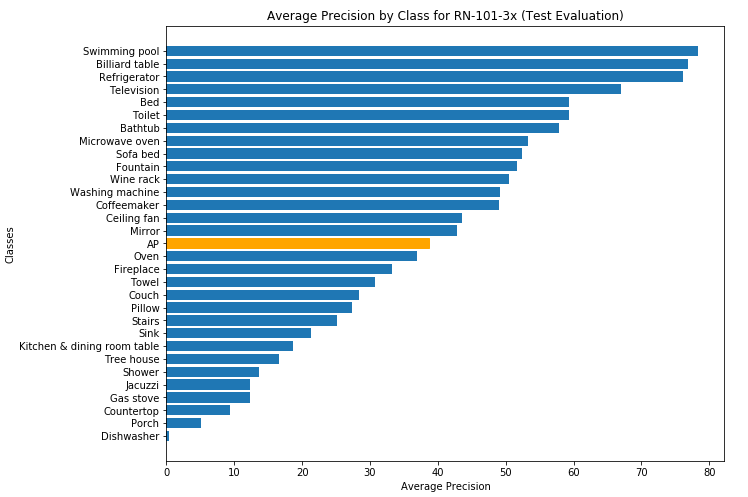
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Training Time | AP (%) | AP50 (%) | AP75 (%) |
| R50-FPN-3x | 2.5 hours | 29.035 | 47.212 | 29.823 |
| R101-FPN-3x | 3 hours | 28.943 | 47.169 | 30.348 |
| RN-50-3x | 6 hours | 39.412 | 58.921 | 42.521 |
| RN-101-3x | 12 hours | 38.994 | 58.644 | 41.618 |

**Test Results (~2k images)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Training Time | AP (%) | AP50 (%) | AP75 (%) |
| R50-FPN-3x | 2.5 hours | 30.18 | 48.59 | 29.82 |
| R101-FPN-3x | 3 hours | 28.77 | 46.62 | 31.42 |
| RN-50-3x | 6 hours | 38.49 | 58.92 | 45.52 |
| RN-101-3x | 12 hours | 38.99 | 58.64 | 41.62 |

**Best Performing Model:** RN-101-3x (We will use this model for deployment).

Achieved an AP of 38.99% which is still far from the Airbnb Google AutoML model (AP: 68%), but given the limited resources and significantly less data points, it is decent.

**Breakdown of AP by Class:**

**Project References:**

* <https://medium.com/airbnb-engineering/amenity-detection-and-beyond-new-frontiers-of-computer-vision-at-airbnb-144a4441b72e>
* <https://github.com/mrdbourke/airbnb-amenity-detection>

**Streamlit**

**To Run Streamlit App:** streamlit run xxx.py

**Docker**

* Create a new folder called docker to keep everything clean
* Cd into that directory
* Build Images (name is mystapp:latest): docker build -t mystapp:latest .
* Build Container (map to port 8080): docker run -it -p 8080:8080 mystapp:latest
* Access app with: <http://192.168.99.100:8080/>
* Dig into a file system of an already run container: docker exec -t -i mycontainer /bin/bash
* Install VIM into docker container:
  + apt-get update
  + apt-get install vim
* Stop container: docker stop containerID
* View all container: docker ps -a
* View all images: docker images
* Remove all containers: docker rm $(docker ps -aq)
* Remove all images: docker rmi $(docker images -q)

**Deploy Docker to GCP**

* <https://www.youtube.com/watch?v=03KgXhg-voY>
* Create app.yaml file
* Ensure that gcloud sdk is installed in local file system
* List of all projects: gcloud projects list
* Look at current project: gcloud config get-value project
* Change to project: gcloud config set project projectID
* Deploy: gcloud app deploy

**Applink:** <https://household-amenity-detection.uk.r.appspot.com/> (has been taken down)

**What could be improved?**

* Performance (Currently at mAP: 39%, Target: 50%)
  + More Training data (Currently at ~25k images)
  + More computational power (Using Google Colab cut off at 12 hours)
  + Imbalanced classes (Some classes only have ~200 images e.g. Bathtub)
* App interface (Streamlit) to be mobile compatible
* Cost too much to be hosted at GCP