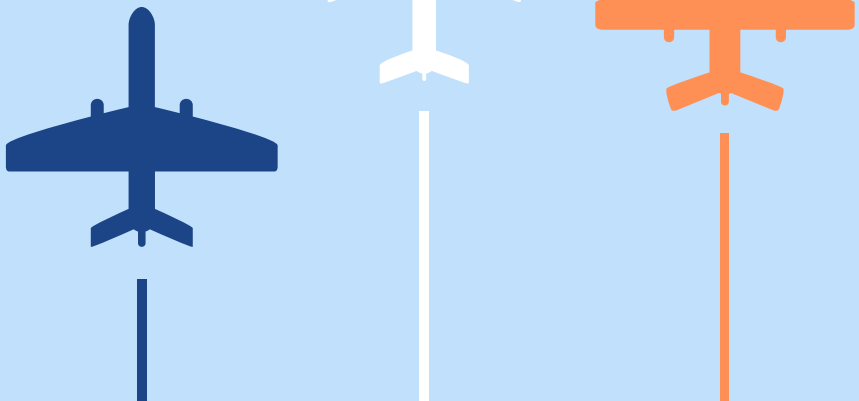
Training and Analyzing Top Performing

Plane Detection
Object Recognition models
across **PyTorch** and **Keras**



December 8, 2020

Airplane detection: a business and security problem

Airports have radar data with callsigns and registration info, but they rely on radio to verify accuracy. What about *deep learning-based verification*?

This amounts to an *anomaly detection* problem via *classification*:

- **Airplane “model family” detection using color cameras around the airport.**
- **Situational awareness systems onboard for use during taxiing and on the runway.**
  (Airbus requested something akin to this while I was at NVIDIA, but more for pilot situational awareness.)

→

?

Technical problem

- In some ways this is a **similar** problem to **classification of animal breeds**
- However, one difference is that “**structure**” of an aircraft is a **differentiator**

- It gets *harder*: Different manufacturers can produce the same design under different names
- But also *easier*: Aircraft are rigid objects

Notable feature differences:

- Jets versus props
- Window shapes
- Door locations
- Ratio of length to diameter/height

Dataset

Hosted by Oxford, the FGVC challenge provides an dataset of “aircraft” images and labels:

- 3 designated classes of JPG images of varying dimensions:
- 3333 images each of `train`, `test`, and `val`

In order to most simply have a greater proportion in ‘train’ rather than ‘test,’
I used the ‘`trainval`’ set of 6666 images to train, and the ‘`val`’ set of 3333 images to test

Label Attribute	Example	Count
Manufacturer	Boeing	41
Family	Boeing 737	70
Variant	Boeing 737-700 series	102
Model	Boeing 737-76J	Not used*

*visually indistinguishable

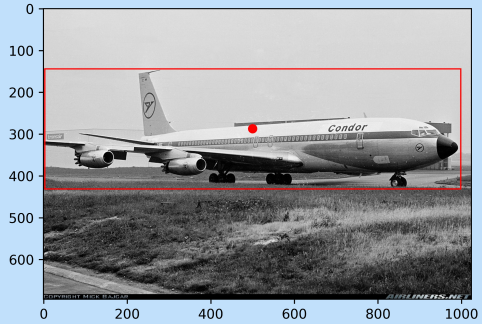
Observation: as this dataset emerged in 2012, it ships only with MATLAB code!

Preprocessing

In order to use this dataset, ostensibly designed for VGG, I needed to:

- Convert monolithic label text files to individual per-image labels with bounding boxes
- Convert pixel-accurate coordinates for bounding boxes to floats
- Move image files to train and val directories based on listing in a text index
- Batch remove bottom pixel rows to prevent “peeking” by photographer attribution

(label file manipulation via Pandas was essential, and PyPlot helped verify my transformations.)



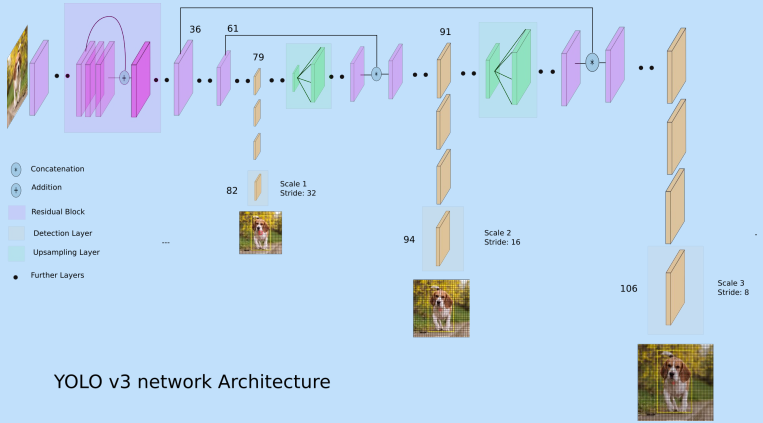
Conceptually, how object detection works

Labels must come with a 4-dimensional vector, either:

- [xmin, ymin, xmax, ymax]
- [x_center, y_center, width, height]

Can be pixel integers or floating point numbers, depending on model/dataset.

Typically each image has a matching “bounding box” defined in its paired text file. In the case of the aircraft dataset, all bounding boxes were enumerated in a single list I converted in Pandas.



YOLO v3 network Architecture

Credit: Ayoosh Kathuria on TDS

Progress over time: YOLOv3 versus YOLOv5

YOLOv3:

Training time: 7h45m for 100 epochs and batch size 48 on large/standard-sized model with COCO-derived pre-trained weights.

YOLOv5:

Training time: 7h35m for 100 epochs and batch size 32 on medium-sized model with COCO-derived pre-trained weights.

NVIDIA T4 for experimentation
(jupyter.brw.ai:8888)

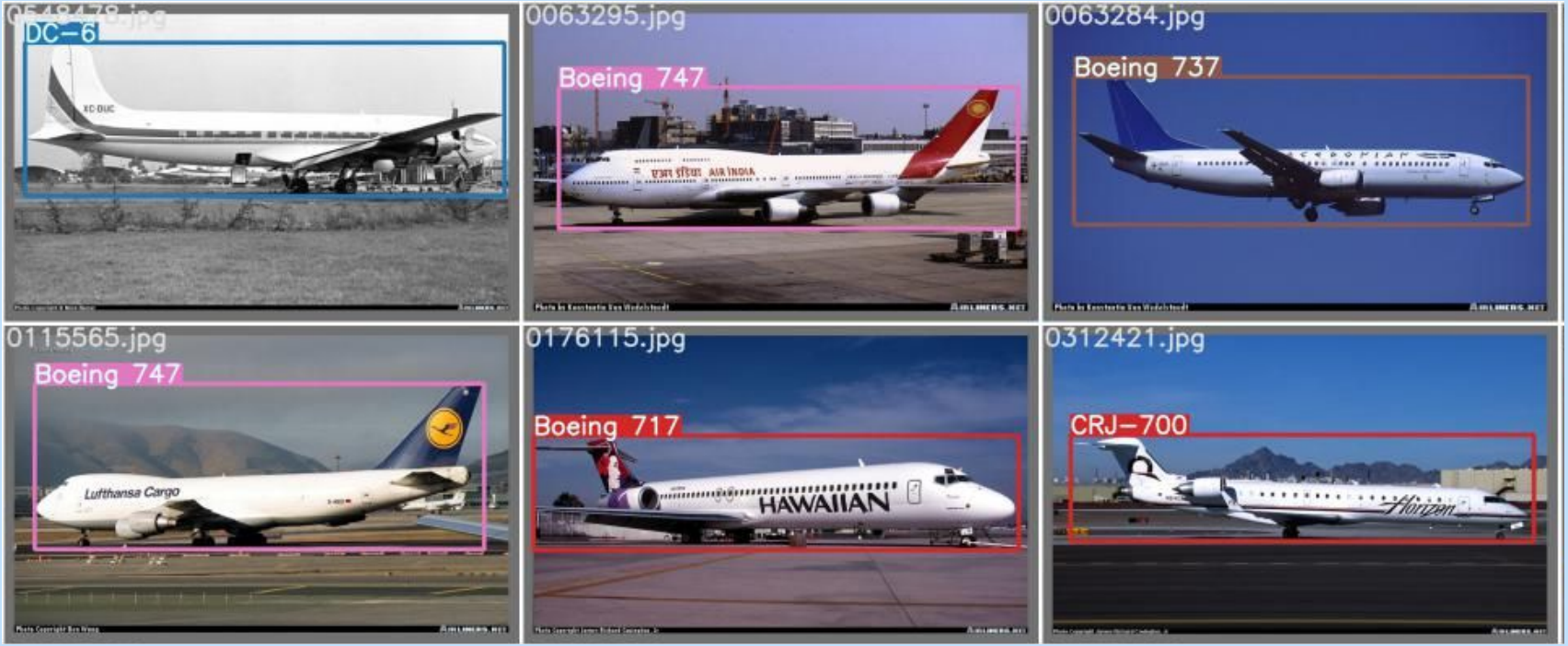
NVIDIA V100 for overnight training
(jupyter-hub.brw.ai)

Google Cloud Platform

Compute Engine

For both models, a batch size of 64 saturated the Tesla V100's 16 GB of HBM2 memory.

Inference on test data (YOLOv3):



Precision: 0.8317 Recall: 0.9508 Mean-average-precision: 0.94 Top 5 0.8614 Top 1

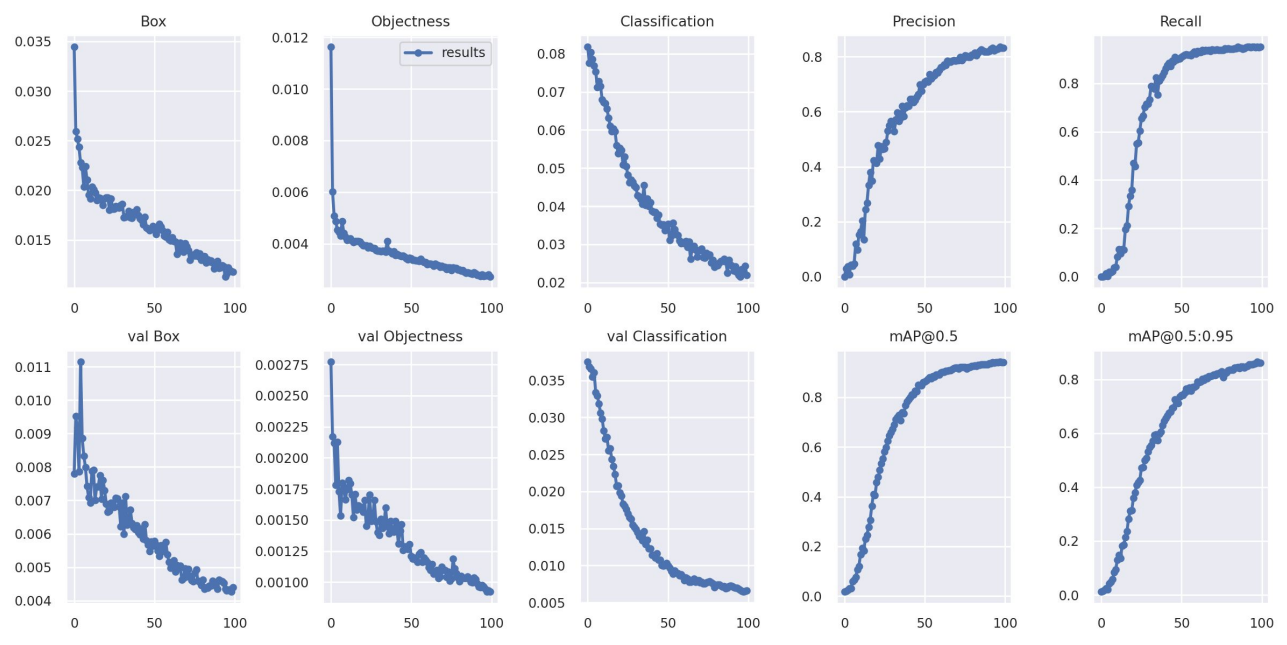
Inference on test data (YOLOv5):

Precision: 0.7675 Recall: 0.9594 Mean-average-precision: 0.9475 Top 5 0.8901 Top 1

Again, no need for early stopping:

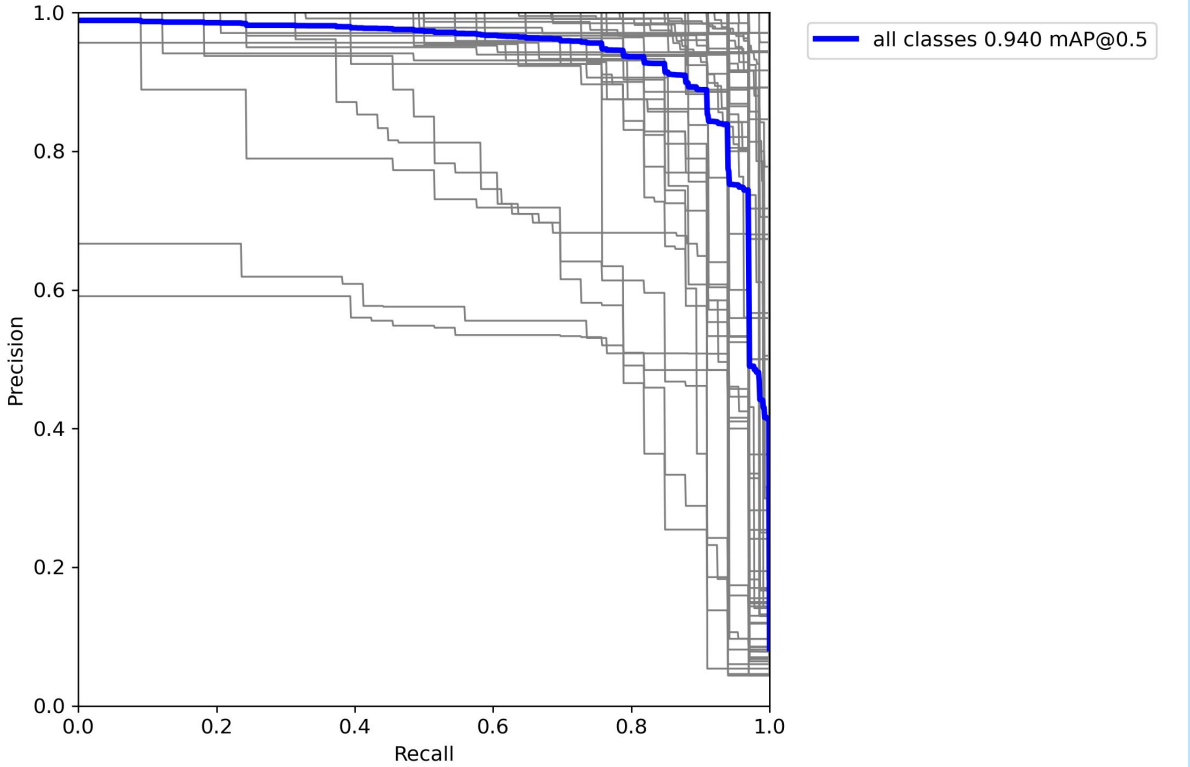
Training

Validation



There may have even been value in further training.
Training curves are slightly smoother than with YOLOv5

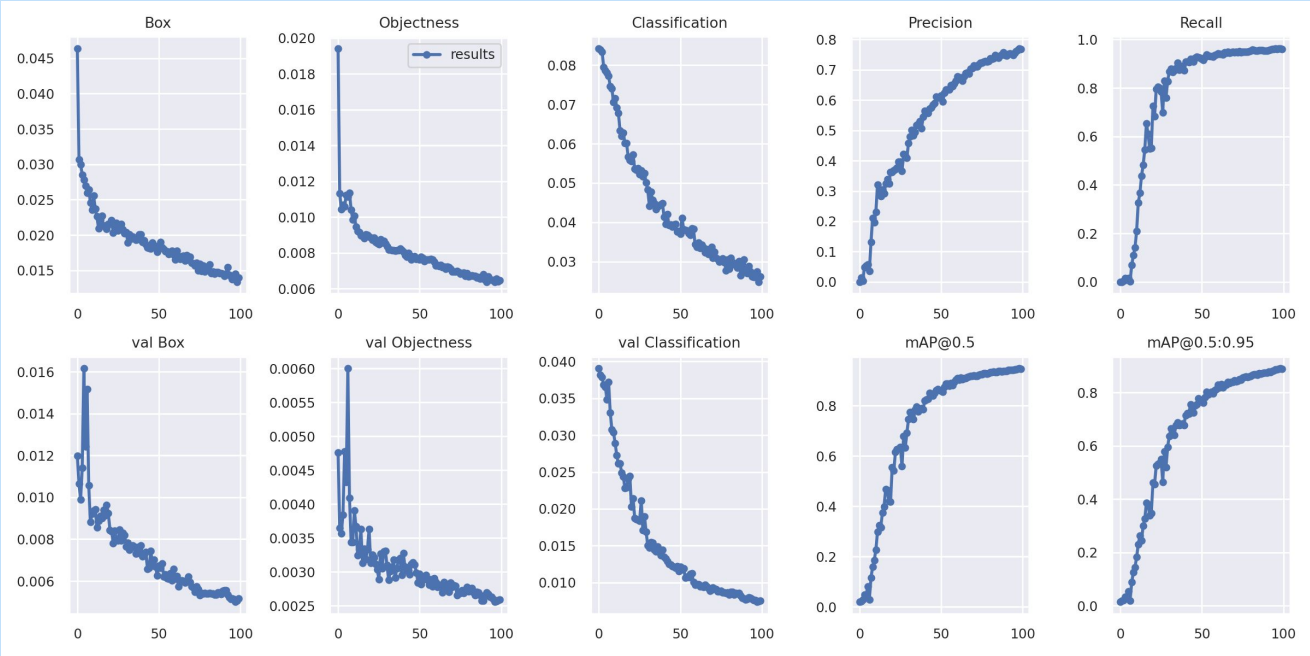
Precision-recall curve for 70 classes (YOLOv3):



No need for early stopping:

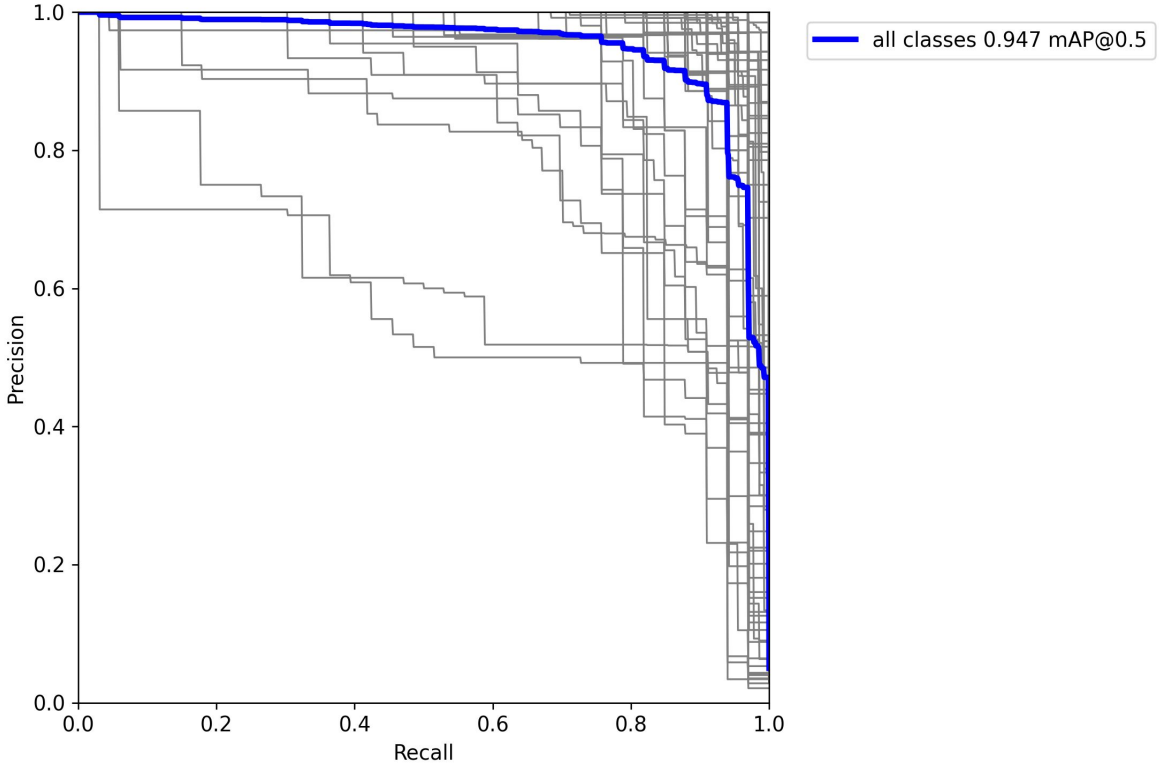
Training

Validation



There may have even been value in further training.

Precision-recall curve for 70 classes (YOLOv5):



Future work

- EfficientDet or EfficientNet versus ResNet, RetinaNet
  ("state of the art" when I joined Google, prior to AmoebaNet/NASnet, anyway)
- Attempt workflow on Cloud TPU
  (if it's priced competitively with the V100; GPUs are expensive enough)
- Weights & Biases logging (done); TensorBoard (not done, no access to port)
- SigOpt hyperparameter optimization
- Flask app with classification samples and file upload
- Re-do precision recall plots so that mouseover selects the class name

Weights & Biases

Questions?

You can download this presentation and the code for this project at:

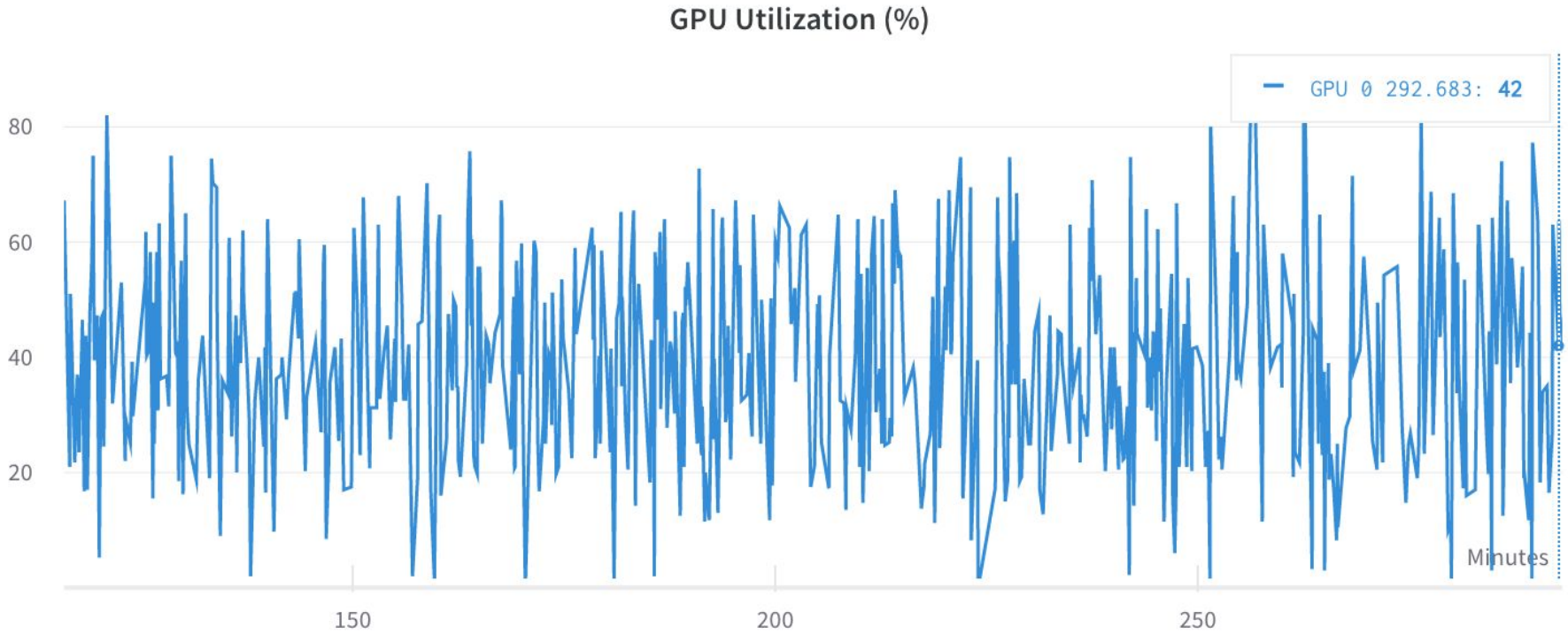
https://github.com/brwillia/metis-aircraft-proj-5

Feel free to keep in touch at:

b@brw.ai

https://www.linkedin.com/in/barwi/

GPU is underutilized much of the time



Per Weights and Biases, training YOLOv3.

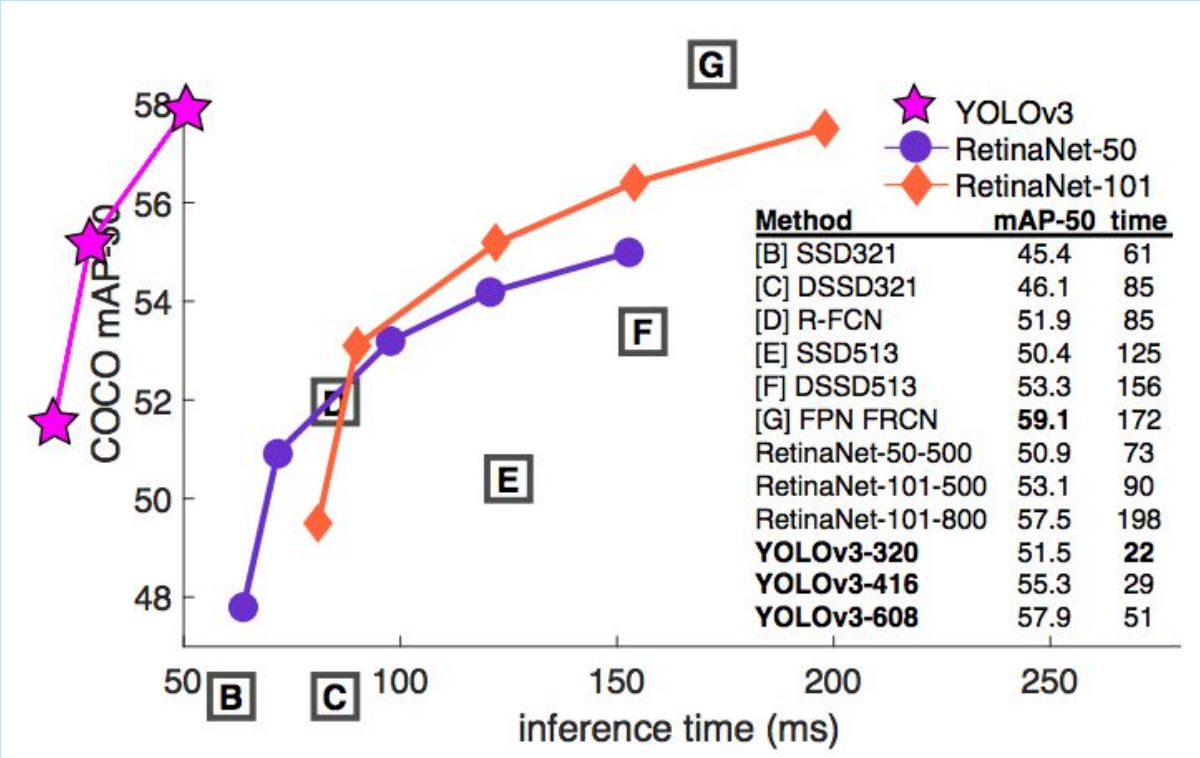
What I learned

- JupyterHub supports multiple users
  - One system-wide conda base, with overlays per-user
  - Sometimes causes incompatibility errors, requiring system-wide override
  - Crashes when training complete, which is nerve-wracking
- V100s are semi-affordable now on GCP
- Don’t assume your translation from one label format to another is correct without visualizing it in Image or PyPlot first
- Some models/approaches really do require old versions of TensorFlow
- Batch sizes don’t have to be powers of 2 (i.e., 2^n)
- Memory leaks can crash training due to increased system RAM usage, rather than GPU VRAM usage (which was stable)

What I learned (continued)

- Reverting to an early (1.x) version of TensorFlow if you've already created your conda environment around a recent version of PyTorch simply won't end well (massive non-resolvable dependency conflicts)

Comparison of competitive performance in the literature



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

An alternative family: Inception and EfficientDet/Net

MaskRCNN and segmentation