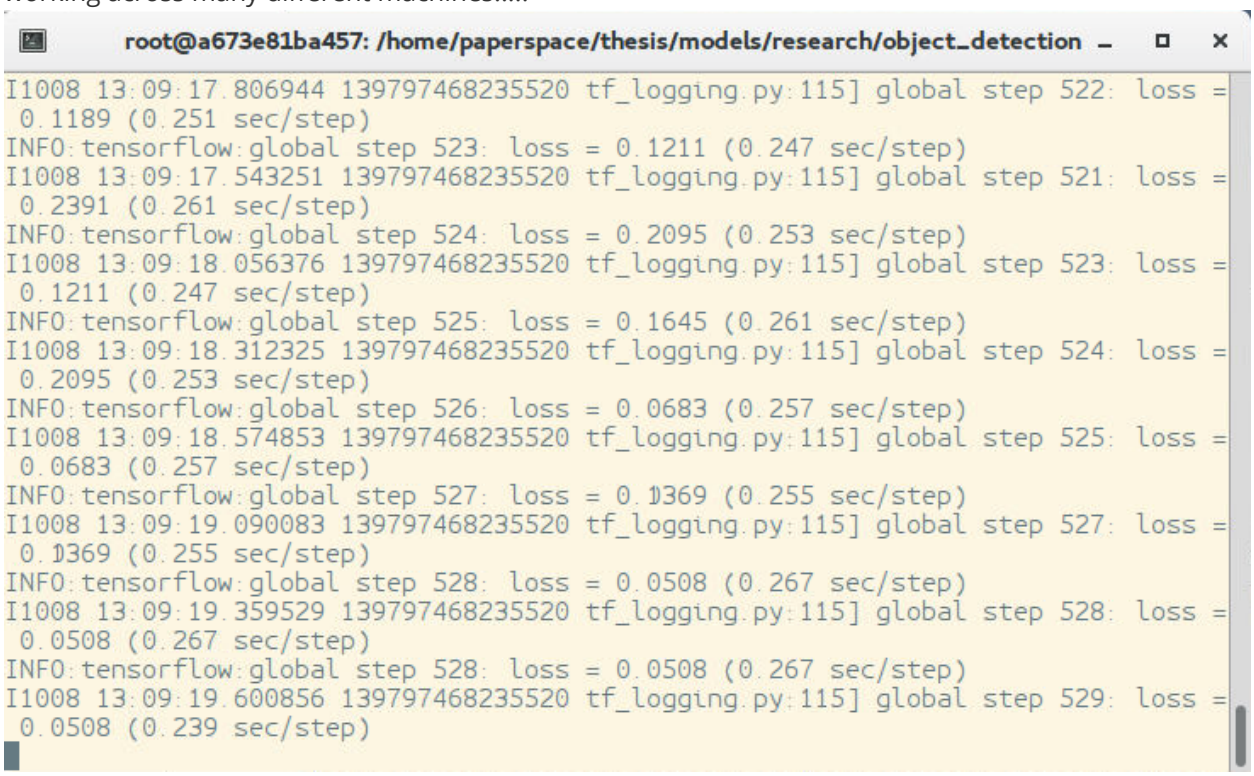
Week 5

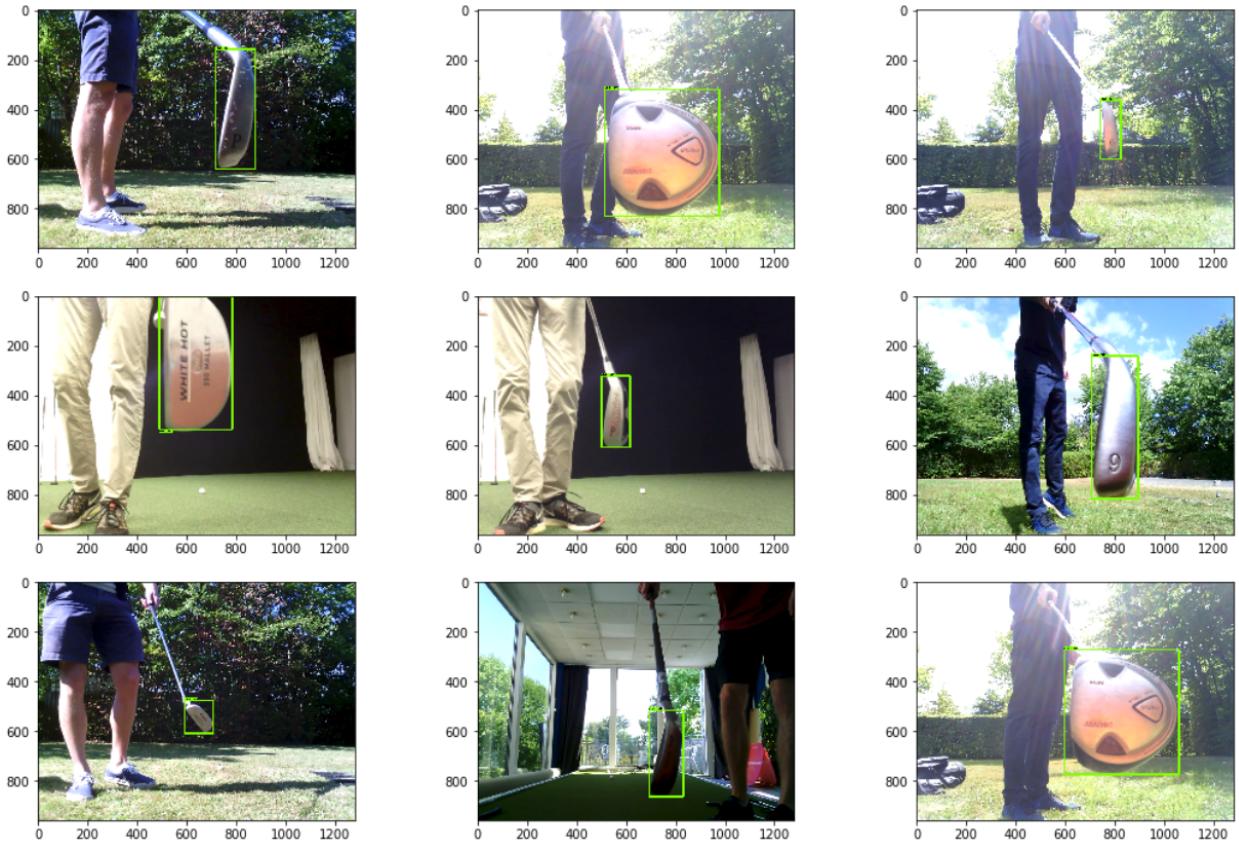
What has been done this week

This week I have been working more on the detection model, making it annotate new and unseen data and getting those annotations into BeaverDam. Also making the script run remotely with docker and locally which is a huge pain saver.

- [x] Get validation script to run and validate the results on the test set
- [x] Train script starter (bash) from within docker
- [x] Docker: Unified container across everything - no more fiddling with keep environment up to date and working across many different machines!!!!!

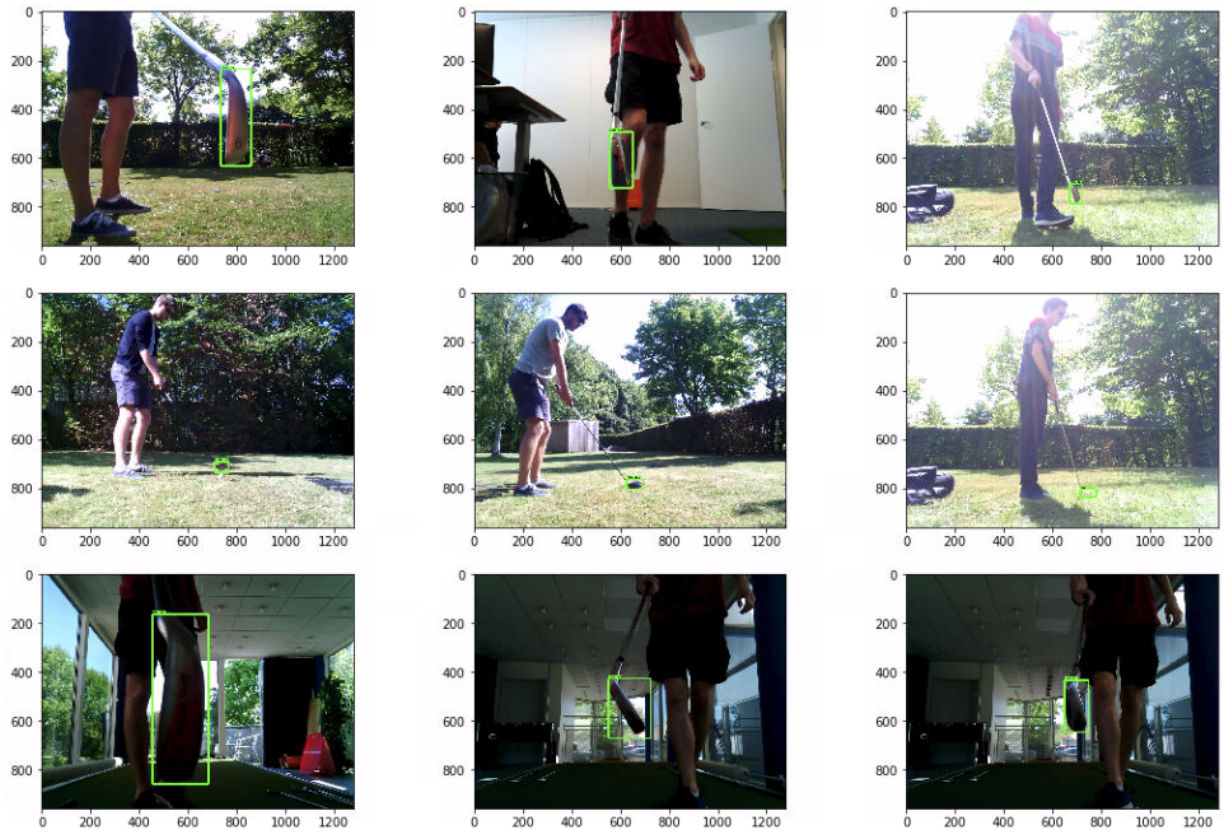


- [x] Get better detection model **It runs super well now on never before seen data (rcnn model):**



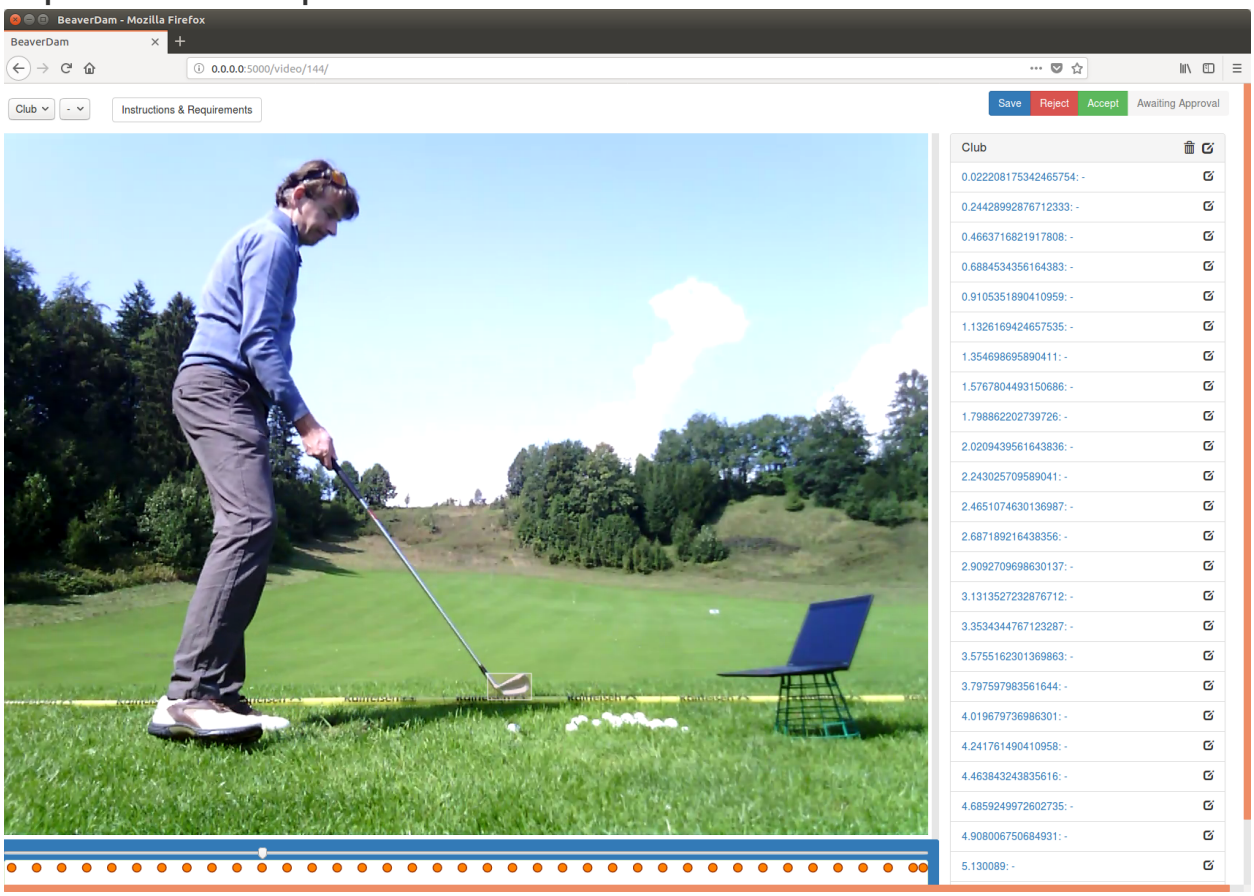
- [x] Train model on full video data on paperspace (cloud solution) - This just means it also trains on the first part of the video i.e the presentation part:

```
../../../data/extracted_frames/test/iron6-20180817_JMB_outdoor_sun_ahead-frame_280.png
../../../data/extracted_frames/test/iron5_1-20180727_JHH_in_the_office-frame_210.png
../../../data/extracted_frames/test/iron5_3-20180726_JJH_outside-frame_160.png
../../../data/extracted_frames/test/wood-20180817_JMB_outdoor_sun_left-frame_60.png
../../../data/extracted_frames/test/driver_1-20180817_JMB_outdoor_sun_right-frame_30.png
../../../data/extracted_frames/test/iron5_3-20180726_JJH_outside-frame_50.png
../../../data/extracted_frames/test/iron7_3-20180727_JJH_upstairs-frame_300.png
../../../data/extracted_frames/test/iron9_1-20180727_JJH_upstairs-frame_340.png
../../../data/extracted_frames/test/iron9_1-20180727_JJH_upstairs-frame_360.png
```



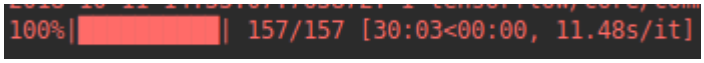
- [x] Train model with reduced bounding boxes (50) for the first step.

- [x] Get annotations from RCNN model, put them into the beaverDam database along with the videos and check that the script works **This works really nicely. Now I can record new data and get a lot of help in the annotation process from the neural network.**



Doesn't always work tho:

Computation time: 157 videos -> 30 minutes | fraction of frames processed: 0.07 | 11 sec / video



Computation time: 157 videos -> 60 minutes | fraction of frames processed: 0.15 | 23 sec / video

Questions / difficulties

I'm still considering the best way of tackling the classification problem. Standard classification with neural networks is quite limited because of the data and the variation. Another approach would be to encode the image data and match that encoding with a database. We have many great face detection networks (e.g FaceNet by google), which might be a basis to start from. These have a triplet loss and the facenet model encodes a face into a 128 dimensional vector which can then be matched with a database. Maybe this is an interesting direction.

Literature

Speed/accuracy trads-offs for modern convolutional object detectors (328 cites) - Huang et al

Paper examines **SSD, Faster RCNN R-FCN** (regional-bassed fully convolutional networks) with 5 different backbones: **VGG-16, Resnet-101, Inception V2, Inception V3, Inception Resnet (v2), MobileNet**

Super nice paper

> Our findings show that using fewer proposals for Faster R-CNN can speed it up significantly without a big loss in accuracy, making it competitive with its faster cousins, SSD and RFCN. We show that SSDs performance is less sensitive to the quality of the fea- ture extractor than Faster R-CNN and R-FCN. And we identify sweet spots on the accuracy/speed trade-off curve where gains in accuracy are only possible by sac- rificing speed (within the family of detectors presented here).

> In- spired by recent successes on image classification [20], the R-CNN method took the straightforward approach of crop- ping externally computed box proposals out of an input im- age and running a neural net classifier on these crops. This approach can be expensive however because many crops are necessary, leading to significant duplicated computation from overlapping crops. Fast R-CNN [10] alleviated this problem by pushing the entire image once through a feature extractor then cropping from an intermediate layer so that crops share the computation load of feature extraction.

we now describe. For each anchor a, we first find the best matching groundtruth box b (if one exists). If such a match can be found, we call a a "positive anchor", and assign it (1) a class label $y_a \in \{1 \dots K\}$ and (2) a vector encoding of box b with respect to anchor a (called the box encoding $\phi(b_a; a)$). If no match is found, we call a a "negative anchor" and we set the class label to be $y_a = 0$. If for the anchor a we predict box encoding $f_{loc}(\mathcal{I}; a, \theta)$ and corresponding class $f_{cls}(\mathcal{I}; a, \theta)$, where $\mathcal{I}$ is the image and θ the model parameters, then the loss for a is measured as a weighted sum of a location-based loss and a classification loss:

$$\mathcal{L}(a, \mathcal{I}; \theta) = \alpha \cdot \mathbb{1}[a \text{ is positive}] \cdot \ell_{loc}(\phi(b_a; a) - f_{loc}(\mathcal{I}; a, \theta)) + \beta \cdot \ell_{cls}(y_a, f_{cls}(\mathcal{I}; a, \theta)), \quad (1)$$

where α, β are weights balancing localization and classification losses. To train the model, Equation 1 is averaged over anchors and minimized with respect to parameters θ.

The input size configuration for SSD might lead to problem in my domain (golf club head are usually very long on one edge):

3.3. Input size configuration.

In Faster R-CNN and R-FCN, models are trained on images scaled to M pixels on the shorter edge whereas in SSD, images are always resized to a fixed shape $M \times M$. We

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

Table 2: Properties of the 6 feature extractors that we use. Top-1 accuracy is the classification accuracy on ImageNet.

Results:

SSD's are fast but quite inaccurate, especially because they are bad for small objects

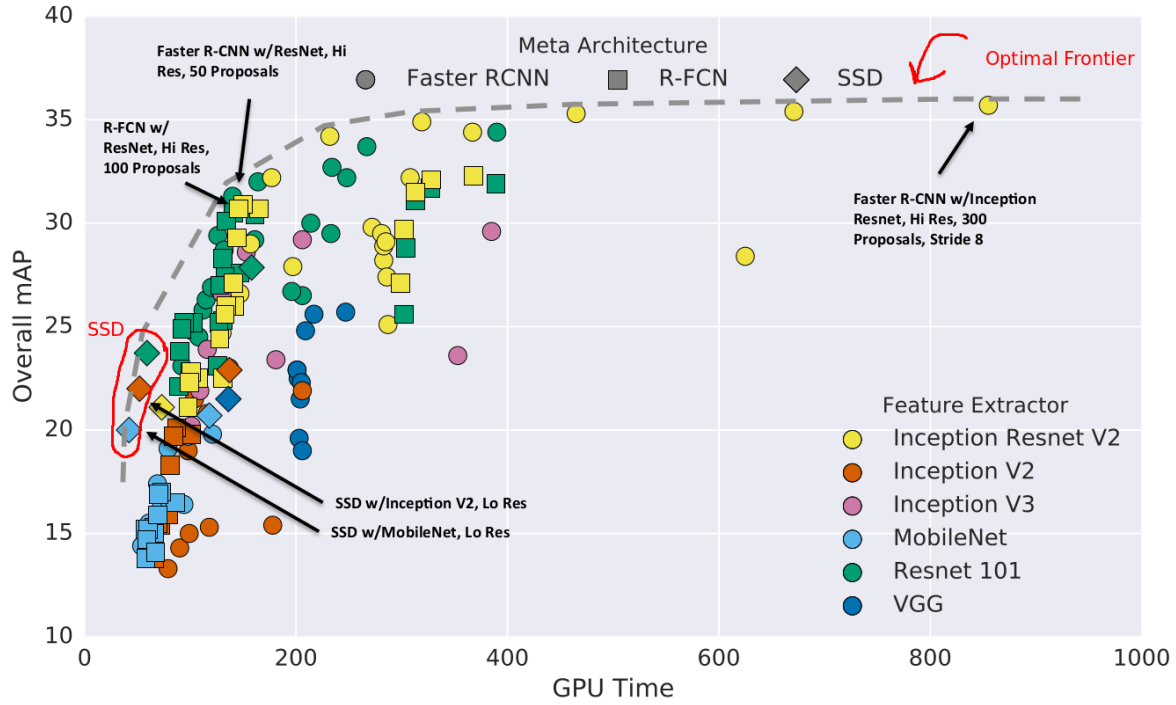
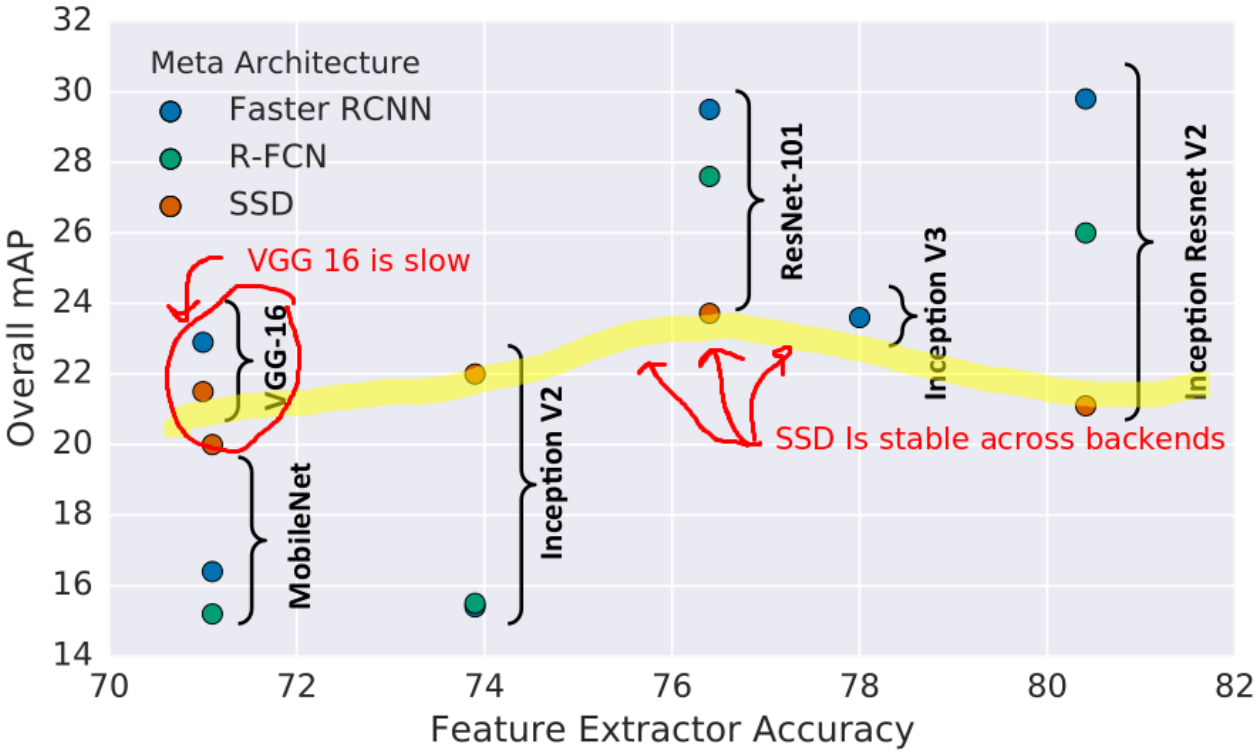


Figure 2: Accuracy vs time, with marker shapes indicating meta-architecture and colors indicating feature extractor. Each (meta-architecture, feature extractor) pair can correspond to multiple points on this plot due to changing input sizes, stride, etc.

Model summary	minival mAP	test-dev mAP
(Fastest) SSD w/MobileNet (Low Resolution)	19.3	18.8
(Fastest) SSD w/Inception V2 (Low Resolution)	22	21.6
(Sweet Spot) Faster R-CNN w/Resnet 101, 100 Proposals	32	31.9
(Sweet Spot) R-FCN w/Resnet 101, 300 Proposals	30.4	30.3
(Most Accurate) Faster R-CNN w/Inception Resnet V2, 300 Proposals	35.7	35.6

Table 3: Test-dev performance of the "critical" points along our optimality frontier.

SSD seems to be able across multiple different backends:



SSD has bad performance for small objects which is a challenge as the golf clubs are often quite small in the picture

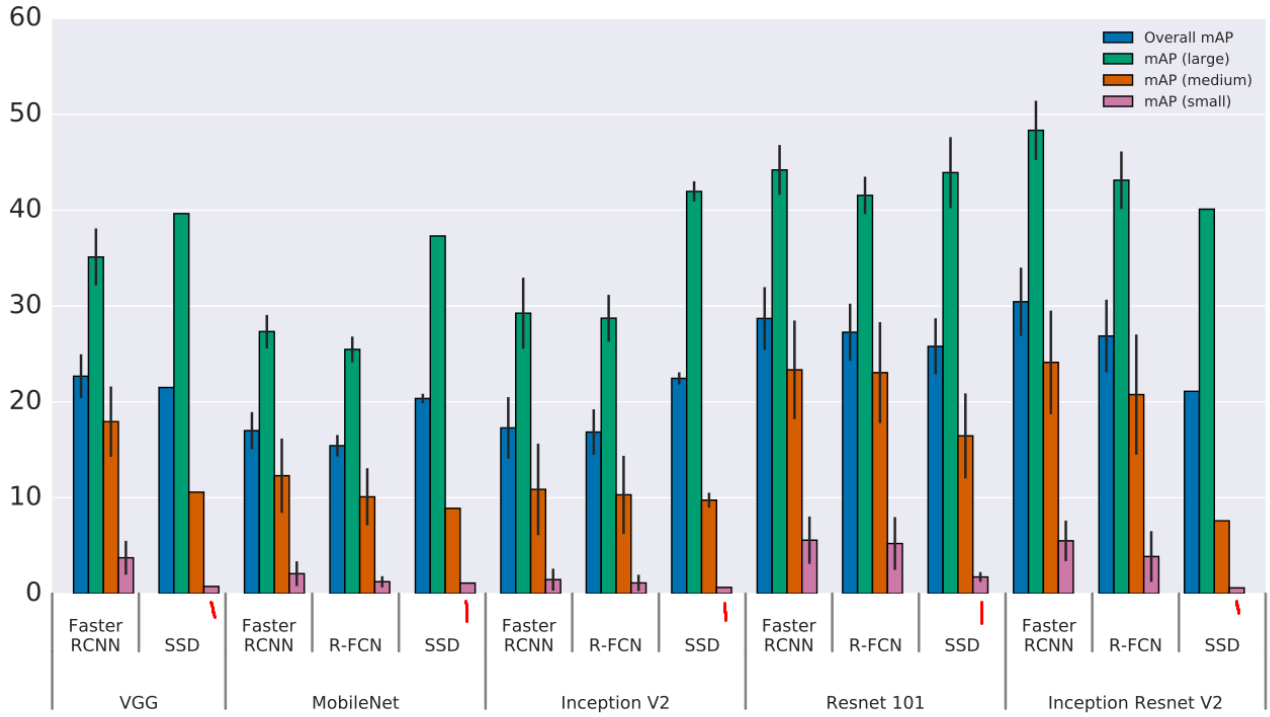


Figure 4: Accuracy stratified by object size, meta-architecture and feature extractor, We fix the image resolution to 300.

4.1.4 The effect of object size.

Figure 4 shows performance for different models on different sizes of objects. Not surprisingly, all methods do much better on large objects. We also see that even though SSD models typically have (very) poor performance on small objects, they are competitive with Faster RCNN and R-FCN on large objects, even outperforming these meta-architectures for the faster and more lightweight feature extractors.

Higher resolutions resolves small objects better

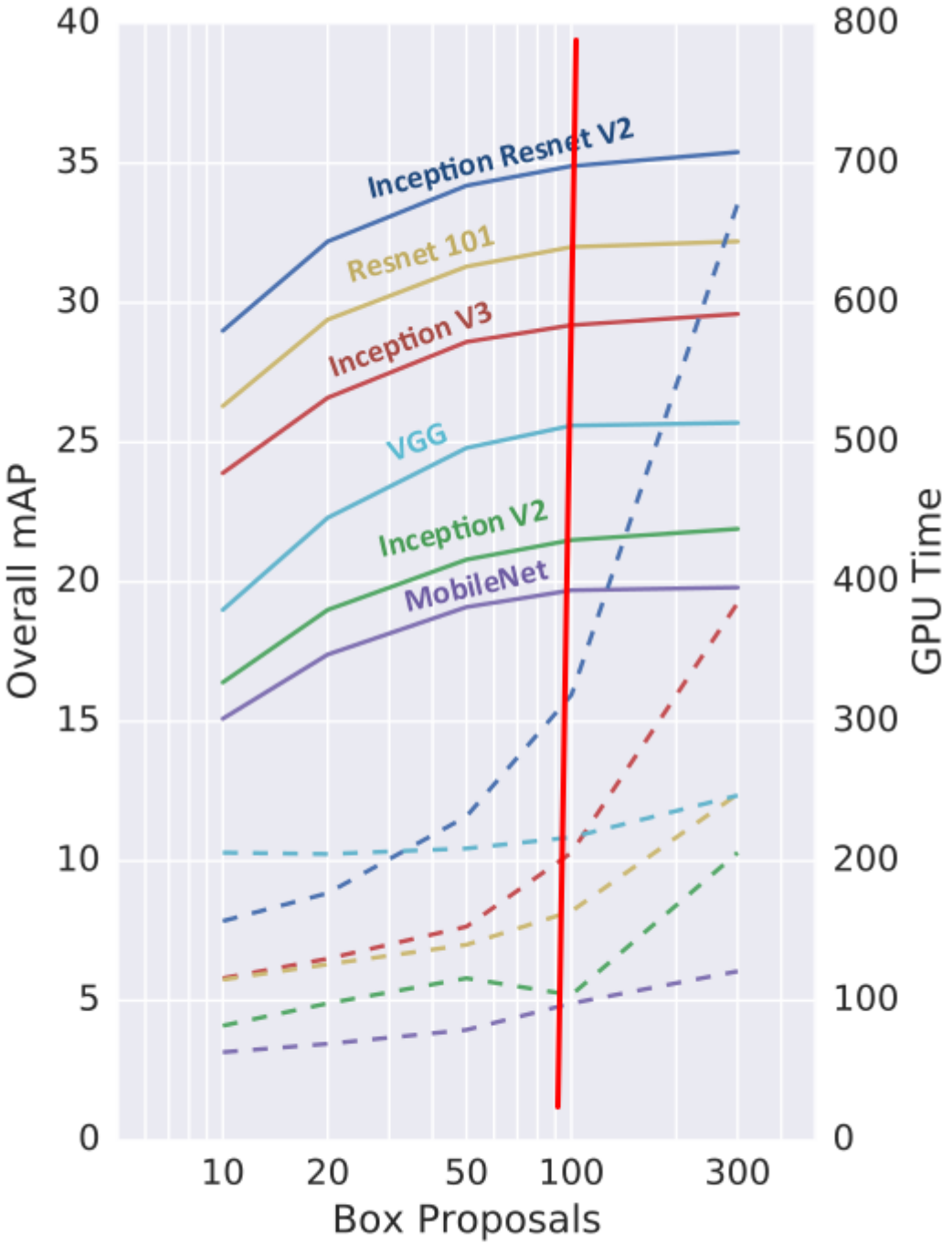
4.1.5 The effect of image size.

It has been observed by other authors that input resolution can significantly impact detection accuracy. From our experiments, we observe that decreasing resolution by a factor of two in both dimensions consistently lowers accuracy (by 15.88% on average) but also reduces inference time by a relative factor of 27.4% on average.

One reason for this effect is that high resolution inputs allow for small objects to be resolved. Figure 5 compares

Reducing the number of proposals (default 300 in detection API) of FRCNN can increase performance a lot:

> We see that Inception Resnet, which has 35.4% mAP with 300 proposals can still have surprisingly high accuracy (29% mAP) with only 10 proposals. The sweet spot is probably at 50 proposals, where we are able to obtain 96% of the accuracy of using 300 proposals while reducing running time by a factor of 3



(a) FRCNN

NOTES:

This paper is from 2016 and the authors acheivede state of the art performance on the COCO object detection that year by making an ensemble of 5 faster RCNN based models based on resnet and inception.

The best networks today seem to use the newer **feature pyramid networks** which I will have to look into.

The paper is made from the main authors of the **tensorflow object detection API**, thus the API is originally created to make this paper and compare the different networks because they fit into the same training scheme. For current state of the art performance, you probably want to look elsewhere for the **FPN** networks.

Status according to project plan

This week should have been "artificial data generation" according to project plan, but I needed to put more work into the detection model and also use it to annotate new data. Quite some progress has been made in this regard, so It's all right. This is a more interesting direction for the project for me anyway.

What to do next week

Take a vacation and spend some time with my father - When I come back, I will hopefully have had enough time to think about, which direction I can take the project so that it will become more interesting.

- [ ] BeaverDam: Make annotation guide
- [ ] Finish up object detection
- [ ] Get Azure cloud solution up and running (trackman has a deal with MS, will get a P100)
- [ ] look into pix2pix to generate night training sets: https://github.com/phillipi/pix2pix
- [ ] Annotate important frames in video to feed into classification step
- [ ] Detection: Improve folder structure