**Weekly report**

This week we focused on implementing the Tensorflow batching system on our data.

**Basic input and output layer, no hidden layers**

*Summary: This network gave a training accuracy of 10% when trained demonstrating it can learn.*

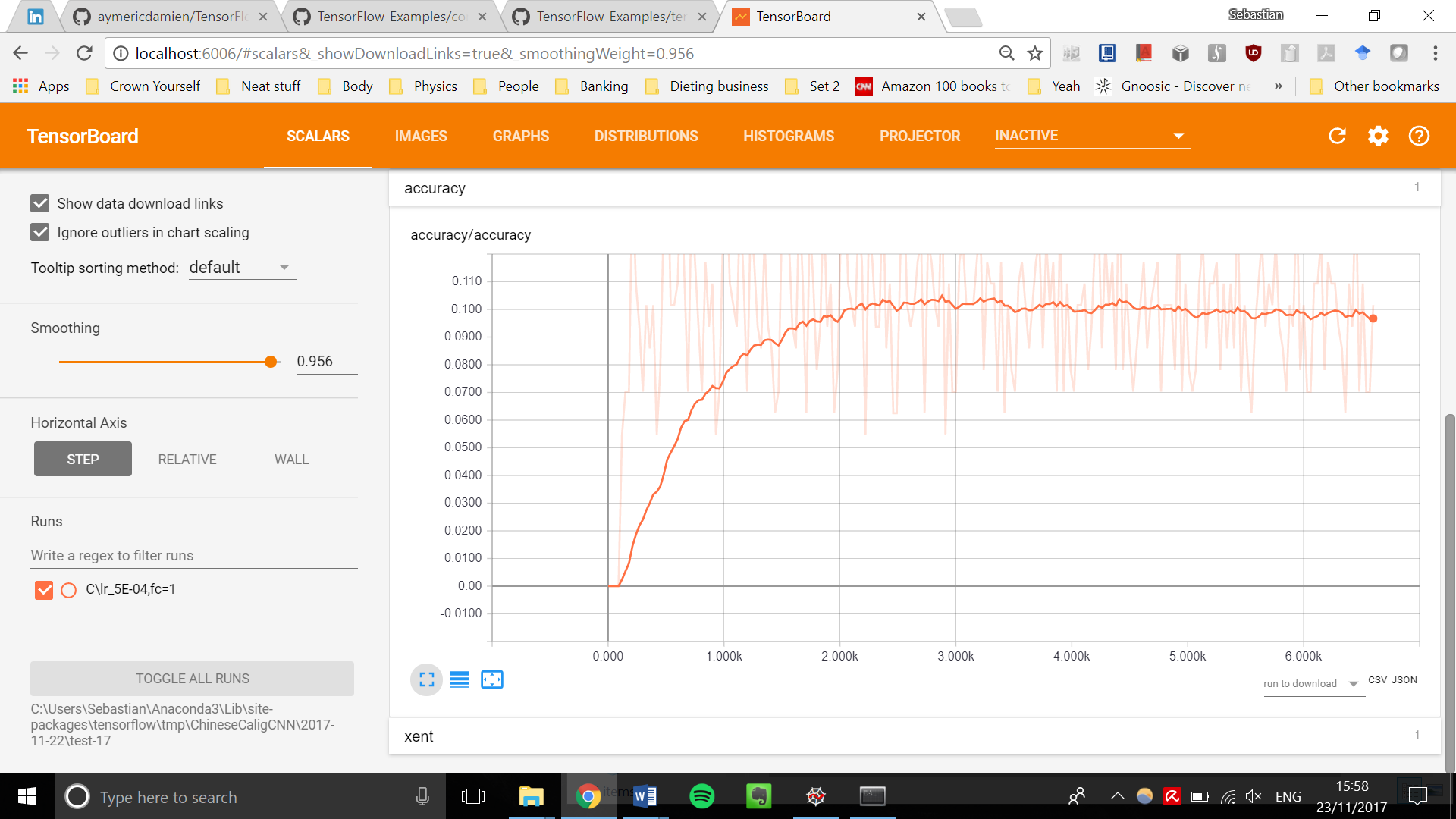
We had a model up and running and had tested many different learning rates and batch sizes, but we were still receiving an accuracy of 0.

We were confused as to why our model was not training at all. To see if it was a problem with just how many files we were trying to run, we ran our system on just two .gnt files to see if we could get some kind of training on such a small dataset. Unfortunately, accuracy remained close to 0 (random fluctuations as the system would sometimes guess correctly).

Since we expected that even with the simplest possible model we would get an accuracy in the single digits (as a percentage), we looked thoroughly through what we were doing and found that the labels we were assigning to each character were incorrect. Since the labels were random, there was no way the network could train.

To address this we tested the networks ability to train itself on just 3000 images for hundreds of epochs. Since each character will have a unique label here (there are 3755 unique characters, and we were only looking at 3000), we wouldn’t face the labelling issue.

Now when we tested the data, we received a training accuracy of 10%! As shown below.



Although this training doesn’t generalise to other data, it does demonstrate the networks ability to physically learn.

**1 hidden Convolution layer**

*Summary: Training accuracy of 0. Possibly due to no drop-out, a learning rate that is too high, or not running over enough iterations (or a combination)*

Next, we tried to add in a convolution layer, again for just 3000 images. Unfortunately, accuracy remained at 0 despite starting from a non-zero value.

In comparison, the simplest network started at a 0 value then after 90 (or 120, in a re-run) iterations, it suddenly increased accuracy, and kept increasing until it reached accuracy of ~0.1 around which it fluctuated.

We have several hypotheses why this may be the case:

* A lack of a dropout rate which switches neurons on/off and lets the network search stochastically for a solution
* A too high learning rate, letting the network ‘settle’ too early
* Not run over enough iterations, so the network can’t try enough random node values

**Fixing the labels**

*Summary: fixed the labelling of our characters, can now train the model on multiple .gnt files*

Once we had fixed our labels, we looked through characters again to double-check our labels are matched properly, and we realised just how different the same character could be, when written by different people. For example, this is the same character written by 10 different people.



Figure 1: The same character written by 10 different people.

There is huge variety in it! If we had to identify identical copies amongst 3754 other different characters we would struggle, let alone a system with the intelligence of a worm.

Now we checked our nets (simple, no hidden layer and one hidden convolution layer) on 10 .gnt files but received the same results. The simple net had a training accuracy of 13%, the net with one convolution layer had an accuracy of 0.

**Action points for the next week**

1. Continue working on the network with 1 convolutional layer, test our hypotheses.

2. Train the simple network with no layers and see what **test** accuracy we can achieve.

3. Keep working on a ML implementation for our data >> is there a better method than one-hot vectors?

4. Look into the tensorflow implementation of array manipulation, rather than manipulating the arrays ourselves.