Weave\_convolution 3.0

Inputs:

* x -> real array of size (examples, C, H, W) or (examples, H, W, C) depending on channels being first or last.
* first\_filters -> integer of the number of local filters in the first convolution map
* filter\_size -> integer of dimension of square filter size (filter\_size by filter\_size)
* p\_factor -> integer from (1 -> filter\_size^2 -1) of the number of peripheral filters for each local filter.
* s\_factor -> integer ratio of the number of filters in the second convolution step combining local and peripheral filters.
* gpu -> selection from {“auto”, N} where “auto” sets the optimal number of gpus based on numerb available and other settings, or N, an integer, forcing the code to use up to min(N, number\_of\_gpus\_available).
* pad\_type -> selection of {‘valid’, ‘same’}.
* f\_activation -> activation of local filters
* f\_reg ->
* s\_activation -> activation of second filters

Psuedo- Algorithm:

1. Pad x to appropriate size -> x\_pad
2. Convolve x\_pad with Y filters of size (filter\_size, filter\_size, C) -> x\_conv
   1. Y1 = (1 + p) \* first\_filters
   2. This can be split up over n\_gpus
3. Split x\_conv into local and peripheral sections
   1. x\_conv size is (num\_examples, filter\_size, filter\_size, Y1)
   2. x\_conv\_local = x\_conv[:,:,:,0: first\_filters \_size]
   3. x\_conv\_peri = x\_conv[:,:,:,first\_filters, p\* first\_filters]
4. Weave x\_conv\_local and x\_conv\_peri into x\_conv\_large
   1. Use tf.map\_fn
   2. And 2D index matrix made with np.where
5. Pad x\_conv\_large -> x\_conv\_large\_pad
6. Convolve x\_conv\_large\_pad with Y2 filters of size (filter\_size, filter\_size, first\_filters)
   1. Y2 = s\_factor \* first\_filters