test\_iter: 100 : test on 100 batches every time we test

test\_interval: 100 : test after every 100 training iterations

base\_lr: 0.001 : initial learning rate for SGD. We solve using SGD algo. Other choices are Adam and RMSprop

lr\_policy: "step" : defines how learning rate changes during training. Here we step the learning rate by a factor gamma every stepsize iterration

gamma: 0.1

stepsize: 2000

display: 20 : display current training loss and accuracy every 20 iterations

max\_iter: 10000 : # of times to update the net

momentum: 0.9 : Other SGD hyperparameters. Setting a non-zero momentum takes weighted average of the current gradient and previous gradient to make learning

weight\_decay: 0.0005 more stable . L2 weight decay regularizes learning, to help prevent the model from overfitting

snapshot: 1000 : snapshots are files used to store network we have trained. We will snapshot every 1000 iteration

snapshot\_prefix: "models/arpit\_alex/caffe\_alexnet\_train"

solver\_mode: CPU

data augmentation to increase the size of your dataset. ( you can at least double or quadruple it)

I know that a smaller batchsize can get you out of local optimas, and thus helps in this regard.but at the same time, a bigger batch-size creates a more stable gradient.

caffenet : A variant of Alexnet

CAffe

Alexnet

train.prototxt

Data layer for test and train

conv1 -> relu1 -> norm1 -> pool1 -> conv2 -> relu2 -> norm2 -> pool2 -> conv3 -> relu3 -> conv4 -> relu4 -> conv5 -> relu5 -> pool5 -> fc6 (innerproduct) -> relu6 -> drop6 (dropout) -> fc7 -> relu7 -> drop7 -> fc8 -> accuracy -> loss

??

Change in Flicker Net

conv2 - bias\_filler: value = 1

conv4 - bias\_filler: value = 1

conv5 - bias\_filler: value = 1

fc6 - bias\_filler: value = 1

fc7 - bias\_filler: value = 1

deploy.prototxt

data layer (not defined for train or test)

conv1 -> relu1 -> norm1 -> pool1 -> conv2 -> relu2 -> norm2 -> pool2 -> conv3 -> relu3 -> conv4 -> relu4 -> conv5 -> relu5 -> pool5 -> fc6 -> relu6 -> drop6 -> fc7 -> relu7 -> drop7 -> fc8 -> prob(softmax)

less learning rate

later layers learn more

decrease the overall learning rate base\_lr in the solver prototxt, but boost the lr\_mult on the newly introduced layer

we set stepsize in the solver to a lower value than if we were training from scratch, since we’re virtually far along in training and therefore want the learning rate to go down faster.

we could also entirely prevent fine-tuning of all layers other than fc8\_flickr by setting their lr\_mult to 0