1 Appendix

1.1 Hyperband search space

To make our entry matrix memory management simple, we fixed batch size as 100 and hence it was not included in our search space. For all the experiments, constant seed was used to get deterministic results.

1.1.1 LeNet

Table 1: Hyper-parameter search space for LeNet network

Hyper-parameter	Scale	Min	Max
Learning rate	log	5e-5	5
Conv1 L2 penalty	log	5e-5	5
Conv2 L2 penalty	log	5e-5	5
Momentum	log	0.001	0.99

We have used caffe's implementation of LeNet network [4] and the details of our search space can be found in table 1. Once a hyper-parameter set is selected, we train a new Lenet (from scratch) using that set for 10000 iterations to report final validation error.

1.1.2 Alexnet

Table 2: Hyper-parameter search space used for CQ

Hyper-parameter	Scale	Min	Max
Learning rate	log	5e-5	5
L2 penalty for all conv layers	log	5e-5	5
FC1 L2 penalty	log	5e-5	5
FC2 L2 penalty	log	5e-5	5

Table 3: Hyper-parameter search space for AlexLRN

Hyper-parameter	Scale	Min	Max
Learning rate	log	5e-5	5
L2 penalty for all conv layers	log	5e-5	5
FC1 L2 penalty	log	5e-5	5
LRN:			
Scale	log	5e-6	5
Power	linear	0.01	3

The details of our search space for CQ network [5], AlexBN [6] and AlexLRN [7] can be found in table 2, 4 and 3 respectively. Once a hyper-parameter set is selected, we train a new CQ, AlexLRN and AlexBN (from scratch) using that selected configuration for 4000, 60000 and 60000 iterations respectively to report final validation error. We have replaced sigmoid with ReLu for AlexBN because we observed that ReLu helps in converging quickly than sigmoid activation function and also found that applying batch normalization after ReLu and pooling layers helps to get better accuracy.

[4]The model specification is available at `https://github.com/BVLC/caffe/blob/master/examples/mnist/lenet_train_test.prototxt`

[5]The model specification is available at `https://github.com/BVLC/caffe/blob/master/examples/cifar10/cifar10_quick_train_test.prototxt`

[6]The model specification is available at `https://github.com/BVLC/caffe/blob/master/examples/cifar10/cifar10_full_sigmoid_train_test_bn.prototxt`

[7]The model specification is available at `https://github.com/BVLC/caffe/blob/master/examples/cifar10/cifar10_full_train_test.prototxt`

Table 4: Hyper-parameter search space used for AlexBN network

Hyper-parameter	Scale	Min	Max
Learning rate	log	5e-5	5
Momentum	log	5e-5	5
L2 penalty for all conv layers	log	5e-5	5
FC1 L2 penalty	log	5e-5	5

1.1.3 MLP

The search space for MLP can be found in Table 5. It has only one hidden, input and output layer with an optional dropout layer which was also used in meprop. Once a hyper-parameter set is selected, we train a new MLP (from scratch) using that selected configuration for 20 epochs.

Table 5: Hyper-parameter search space used in hyperband for MLP

Hyper-parameter	Scale	Min	Max
Hidden neurons	linear	100	8192
Dropout	linear	0	0.999
top-K	linear	0	100

1.1.4 ResNet-18

The search space for ResNet-18[8] can be found in Table 6. We use ADAM optimizer which implements adaptive learning rate and has well known best values of momentum. Hence we did not include the learning rate and momentum in our search space. Once a hyper-parameter set is selected, we train a new ResNet-18 (from scratch) using that selected configuration for 30000 iterations.

Table 6: Hyper-parameter search space used in hyperband for MLP

Hyper-parameter	Scale	Min	Max
L2 penalty for all conv layers	log	5e-5	5
Dropout	linear	0	0.999

1.2 Proof

Total execution time without our technique is given by

$$N(t_f + t_b)$$

Time spent by CNN when our technique is applied consists of 3 parts:

1. time spent in pre-stan interval is
$$p_t(t_f + t_b)$$
2. time spent during calculating E is
$$\frac{N - p_t}{Z + 1}(t_f + t_b + t_e)$$
3. where as the new execution time of CNN after applying STAN is
$$(N - p_t)\frac{Z}{Z + 1}(t_b + (t_f - t_i))$$

So the speedup is given by

[8]The model specification is available at `https://github.com/letr63jd56/STAN/blob/master/stan-cifar10/models/old_resnet_dropout/trainval_bn_relu_conv.prototxt`

$$\frac{N(t_f + t_b)}{p_t(t_f + t_b) + \frac{N-p_t}{Z+1}(t_f + t_b + t_e) + (N - p_t)\frac{Z}{Z+1}(t_b + (t_f - t_i))}$$
$$\frac{N(t_f + t_b)}{N(t_f + t_b) + \frac{N-p_t}{Z+1}t_e - (N - p_t)\frac{Z}{Z+1}t_i}$$
$$\frac{N(t_f + t_b)}{N(t_f + t_b) - \frac{N-p_t}{Z+1}(Zt_i - t_e)}$$

1.3 Unstructured sparsity

From Figure 1, we can see that occurrence of 1 or 2 consecutive zeros is very high in output activation matrix and hence we can conclude that the matrix has unstructured sparsity. If our technique is to be applied on existing BLAS libraries, we need at least 32 consecutive zeros (size of a warp is 32) which is highly unlikely leading to warp divergence. Hence we have written our kernel where instead of a thread handling output of a neuron, a warp handles its computation and thus preventing warp divergence and generating speedup.

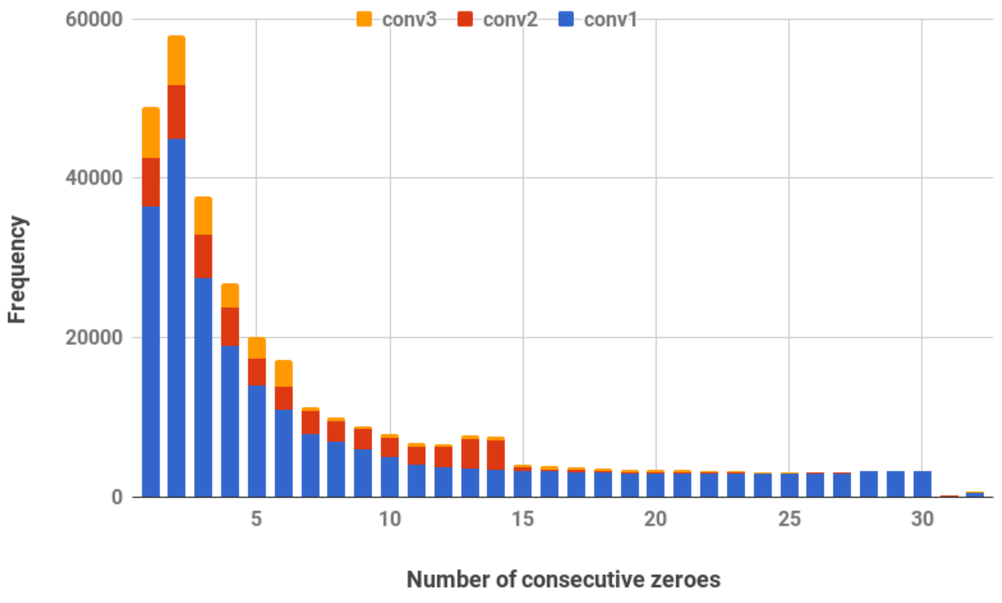


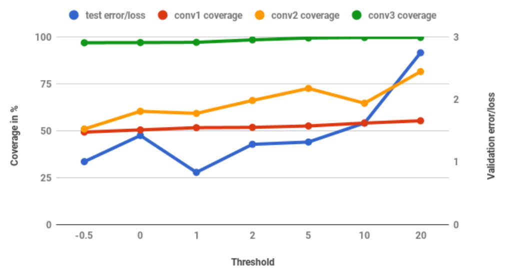
Figure 1: Frequency of consecutive zeros appearing in the output activation matrix across all convolution layers while training CQ CNN for 4000 iterations.

1.4 Effect of varying θ, Z and p_t

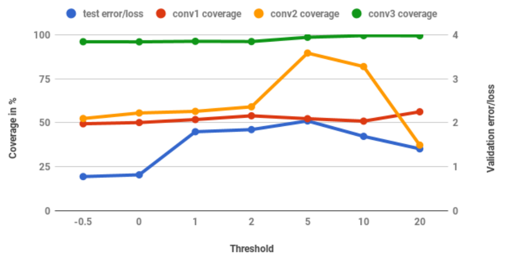
We studied the effect of θ, Z and p_t on validation error (figure 2) and found that as θ increases, both coverage and validation error increases whereas, by increasing p_t or decreasing Z, we achieve better error values. Hence there is a trade-off between speedup and accuracy.

1.5 Validation error change with STAN

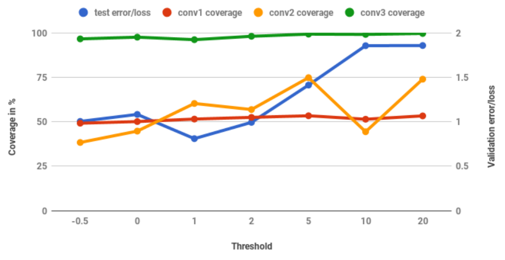
From Figure 3, we see that STAN works better than STAN-lite and when our technique is applied to shallow layers, it acts like regularizer leading to better error than the default/vanilla network.



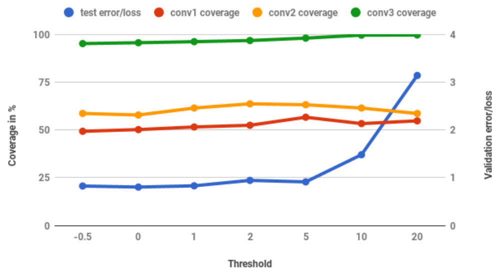
(a)



(b)



(c)



(d)

Figure 2: Effect of various values of θ, Z and p_t on validation error for AlexLRN using CIFAR-10 dataset for 60000 iterations. Here STAN-lite is used and the values for (a)$p\_t = 0$ and $Z = 1$, (b) $p\_t = 1$ and $Z = 1$, (c) $p\_t = 0$ and $Z = 2$, (d) $p\_t = 1$ and $Z = 2$

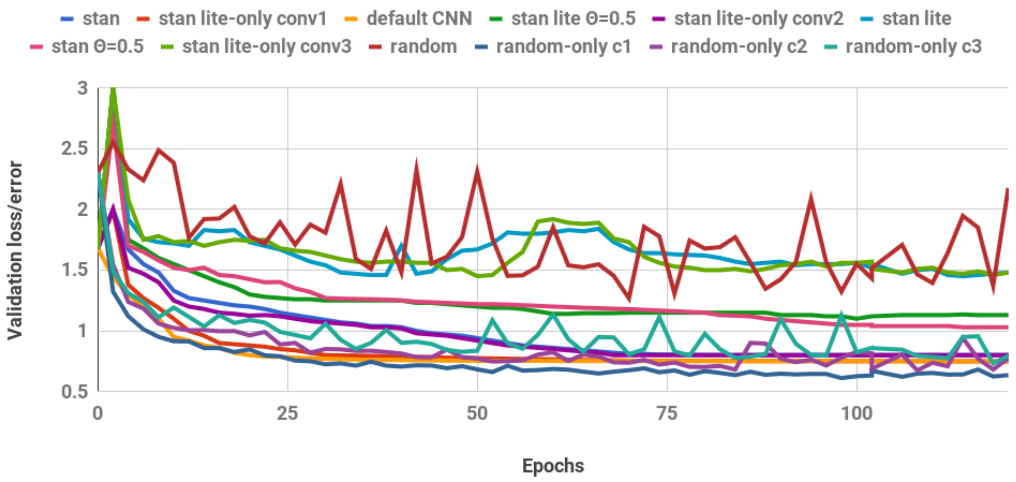


Figure 3: Comparison of validation accuracy while using 2 versions of STAN on all convolution layers for AlexLRN. Here $\theta = 10, p_t = 0, Z = 1$, learning rate=0.001, momentum=0.9 and total epochs=120 or 60000 iterations. The labels containing "only conv" indicates that our technique is applied to only that particular layer and random refers to Convolutional dropout ratio of 0.5.