A ADDITIONAL PLOTS

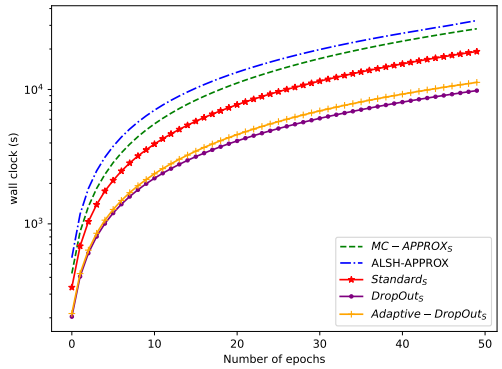

Figure 15: Cumulative training time for each method with 3 hidden layers.

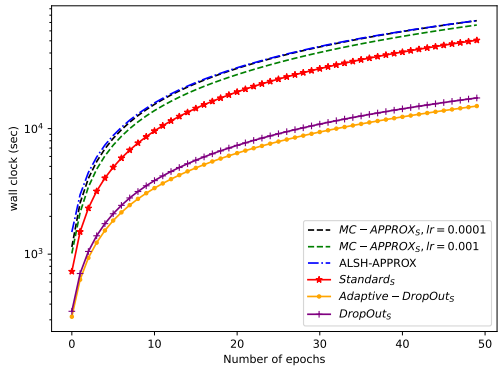

Figure 16: Cumulative training time for each method with 7 hidden layers.

B EXTENSION TO CONVOLUTIONAL NEURAL NETWORKS

The focus of this paper, and the existing work, has mainly been on the fully connected networks (Figure 1), where every node in layer k is connected to every node in layer $k-1$. Another popular architecture is the convolutional neural networks (CNN), which is often used for for grid-like data, such as image data. The key difference of CNNs is their core building block, the *convolutional layer*, which is *not* fully connected. Consider an image to be a $\mu \times \mu$ grid of pixels, passed to a CNN as the input vector with $n = \mu \times \mu$ values, and a kernel of size $n' = \nu \times \nu$, where $\nu \ll \mu$. The kernel can be viewed as a ν by ν patch swept over the image. As a result, each node in a convolutional layer is connected to only n' nodes (pixels that fall inside the patch). It is equivalent of masking out $(n-n')$ values in the weight vector associated with each node in the convolution layer. Subsequently, the weight vector associated with a node will contain only a very small fraction, $\frac{\nu \times \nu}{\mu \times \mu}$, of unmasked values.

Now, let us recall from §4.2 and Figure 3 that the sampling-based techniques either (ALSH-APPROX) select a subset of column and compute the inner-product for them exactly or (MC-APPROX) select all columns but compute the inner-products approximately (by selecting a subset of rows). As a result, MC-APPROX can be extended to convolutional layers by using only the n' edges (rows) from the previous layer to estimate the inner product for each node in the current layer (every column). In particular, MC-APPROX incorporates approximation exclusively within the convolutional layers, while maintaining exact computations in the classifier layers.

Conversely, extending ALSH-APPROX to CNNs is challenging. Recall that ALSH-APPROX uses locality-sensitive hashing to identify the active nodes: the nodes with weight vectors that have maximum inner-product with the input vector. In order to identify the active nodes, ALSH-APPROX *selects rows in* W^k from LSH buckets for a layer k based on their similarity with the input, which is detected by the collision of their hash values. On the other hand, for each node in a convolutional layer, only n' of the rows are unmasked. As a concrete example, in our experiment on the CIFAR-10 dataset, each image is 32×32 pixels and the kernel is 3×3. In this setting, each node in a convolutional layer is connected only to 9 (0.8%) of the 1024 input pixels. The low percentage of unmasked rows for each node in a convolutional layer makes it unlikely that those nodes are among the random selections based on which the hash buckets are constructed.

For a node in a convolutional layer to be identified as active, it should fall into at least one of the L LSH buckets that the input vector falls into. The LSH buckets are constructed using random sign projection as the hash function. That is, each hash function is a random hyperplane that partitions the search space in two halves (positive and negative sides). A set of K such hyperplanes partitions the space into 2^K convex regions, each representing a bucket. As a result, unless the input vector has near-zero values on (a large portion of) the masked rows, it is unlikely that the corresponding node falls into the same bucket as the input vector. Such input vectors (images) are unlikely in practice.

In summary, the hash function that has been built for the entire set of n rows is not an effective near-neighbor index for detecting active nodes for convolutional layers. Alternatively, one can create a separate LSH family, based on the unmasked weights, for each of the n nodes in a convolutional layer; this approach, however, is not practical due to the time and memory overhead for building the hash functions.

Therefore, in order to extend ALSH-APPROX to CNNs, we only consider using LSH for the fully connected (classifier) layers, while maintaining exact computations in convolutional layers. Still, we empirically observed a decline in the model performance in our experiments. Specifically, Table 3 confirms that for a model with 4 convolutional layers with residual blocks (ResNet18) and 2 fully connected layers on the CIFAR-10 dataset, the accuracy dropped to 10.3% (almost as bad as random guessing).