- Discuss VPU/TPU/APU/GPU/FPGA/ASIC memory arcitecture and how it handles matrix sparsity

- Show ineffectivity of pruning on hardware without optimisations for sparse matrices

0.0.1 Memory Allocation

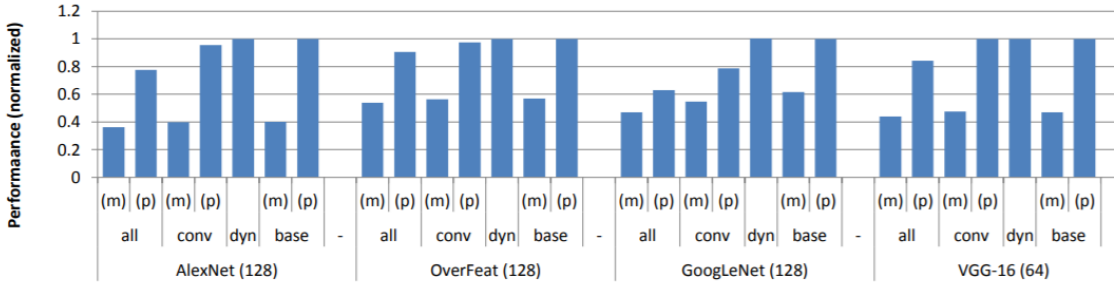

Figure 1: vDNN performance, showing the throughput using various memory allocation strategies.
(Adopted figure from [1])

While designed specifically for training networks that would otherwise be to large for a GPU, the memory manager vDNN proposed in [1] does provide some insight into the importance of memory locality to neural network throughput. Fig. 1 summarizes the performance of vDNN policies compared to a baseline memory management policy (*base*), the vDNN policies include: static policies (denoted as *all* and *conv*) and a dynamic policy (*dyn*). *base* simply loads the full model into the GPU memory consequently providing optimal memory locality. *all* refers to a policy of moving all Xs out of GPU memory, and *conv* only offloads Xs from convolutional layers, Xs are the input matrices to each layer, denoted by the red arrows in Fig. 2. Each of *base*, *conv* and *all* are evaluated using two distinct convolutional algorithms - memory-optimal (m) and performance-optimal (p).

Finally the *dyn* allocation policy chooses (m) and (p) dynamically at runtime. Fig. 1 communicates a significant (58% and 55%) performance loss compared to baseline when no effort is made to optimise the memory locality of parameters in the network

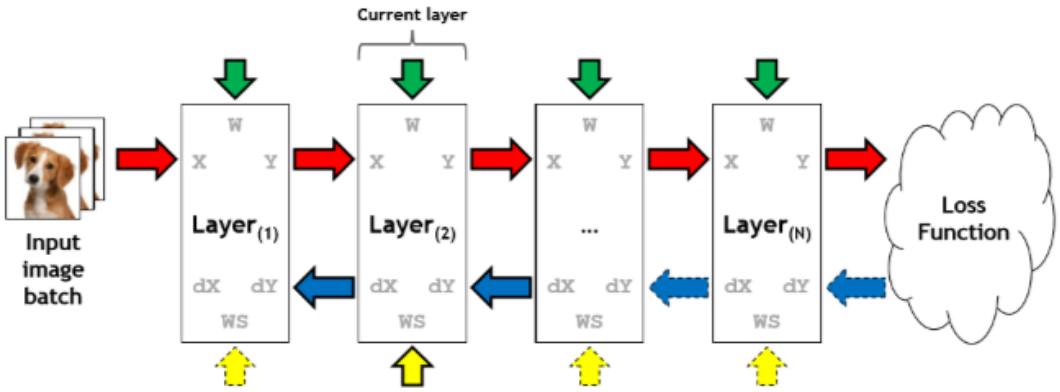

Figure 2: Memory allocations required for linear networks. All green (W) and red (X) arrows are allocated during inference, the blue and yellow arrows are allocated during training.
(Adopted figure from [1])

Justifies the need for compression ... pruning

0.0.2 Memory Access

A significant portion of DNN computation is matrix-vector multiplication, the primary bottleneck of matrix-vector multiplication is memory access [2]. When the cache capacity is insufficient for the input matrix size this bottleneck is expounded, as a consequence memory access is neccessary for every operation since there is no reuse of the input matrix [3]. As observed in 0.0.1 this indicates that compression techniques should alliviate this bottleneck, and while compression doese reduce the total number of operations, the irregular memory access pattern caused by compression hinders effective acceleration see Fig. 3.

Platform	Batch Size	Matrix Type	AlexNet			VGG16			NT-		
			FC6	FC7	FC8	FC6	FC7	FC8	We	Wd	LSTM
CPU (Core i7-5930k)	1	dense	7516.2	6187.1	1134.9	35022.8	5372.8	774.2	605.0	1361.4	470.5
		sparse	3066.5	1282.1	890.5	3774.3	545.1	777.3	261.2	437.4	260.0
	64	dense	318.4	188.9	45.8	1056.0	188.3	45.7	28.7	69.0	28.8
		sparse	1417.6	682.1	407.7	1780.3	274.9	363.1	117.7	176.4	107.4
GPU (Titan X)	1	dense	541.5	243.0	80.5	1467.8	243.0	80.5	65	90.1	51.9
		sparse	134.8	65.8	54.6	167.0	39.8	48.0	17.7	41.1	18.5
	64	dense	19.8	8.9	5.9	53.6	8.9	5.9	3.2	2.3	2.5
		sparse	94.6	51.5	23.2	121.5	24.4	22.0	10.9	11.0	9.0
mGPU (Tegra K1)	1	dense	12437.2	5765.0	2252.1	35427.0	5544.3	2243.1	1316	2565.5	956.9
		sparse	2879.3	1256.5	837.0	4377.2	626.3	745.1	240.6	570.6	315
	64	dense	1663.6	2056.8	298.0	2001.4	2050.7	483.9	87.8	956.3	95.2
		sparse	4003.9	1372.8	576.7	8024.8	660.2	544.1	236.3	187.7	186.5
EIE	**Theoretical Time**		**28.1**	**11.7**	**8.9**	**28.1**	**7.9**	**7.3**	**5.2**	**13.0**	**6.5**
	Actual Time		**30.3**	**12.2**	**9.9**	**34.4**	**8.7**	**8.4**	**8.0**	**13.9**	**7.5**

Figure 3: Wall clock time comparison for sparse and dense matrices between CPU, GPU, mGPU and EIE (an FPGA custom accelerator)
(Adopted figure from [3])

The paper proposing the EIE inference engine [3] describes an excellent technique to exploit the sparity of activations by storing an encoded sparse weight matrix in a variant of compressed sparse column format.