O1	O2	S1	S2	F1	F2
input $3 \times 227 \times 227$		input $3 \times 227 \times 227$		input $3 \times 152 \times 152$	
conv[11, 96, 4, 0]	conv[11, 64, 4, 0]	conv[11, 96, 4, 0]	conv[11, 64, 4, 0]	conv[3, 64, 2, 1]	
pool[3, 2]		pool[3, 2]		pool[2, 2]	
conv[5, 256, 2, 2]		conv[5, 256, 2, 2]		conv[3, 128, 1, 1]	conv[3, 128, 2, 1]
pool[3, 2]		pool[3, 2]		pool[2, 2]	
conv[3, 384, 1, 1]	conv[3, 256, 1, 1]	conv[3, 384, 1, 1]	conv[3, 256, 1, 1]	conv[3, 256, 1, 1]	conv[3, 128, 2, 1]
conv[3, 384, 1, 1]	conv[3, 256, 1, 1]	conv[3, 384, 1, 1]	conv[3, 256, 1, 1]	pool[2, 2]	
conv[3, 256, 1, 1]		conv[3, 256, 1, 1]		conv[3, 256, 1, 1]	conv[3, 256, 2, 1]
pool[3, 2]		pool[3, 2]		pool[2, 2]	
fc[4096]	fc[1024]	fc[2048]	fc[1024]	fc[2048]	fc[1024]
fc[4096]	fc[2048]	fc[2048]	fc[2048]	fc[2048]	fc[1024]
fc[1000]		fc[205]		fc[2622]	

Table 1: Architecture of compact models O1, O2, S1, S2, F1, and F2. The table specifies convolution layers as [kernel size, number of feature maps, stride, padding]; max-pooling layers as [kernel size, stride]; fully-connected layers as [output size].

A Generating Compact Models

Table 1 shows the six compact models used in the paper. We created these models by systematically applying the following operations to publicly available model architecture, such as AlexNet [3] and VGGNet [4]: (a) reduce the number of feature maps, or increase stride size in a convolution layer; (b) reduce the size of a fully-connected layer; (c) merge two convolution layers or a convolution layer and a max-pooling layer into a single layer. The models are unremarkable except that they have fewer operations and layers than original versions and hence run faster, yet achieve high accuracy *when applied to small subsets of the original model's domain*. In fact, how these architectures are derived is orthogonal to the WEG algorithm. We have tested two fully automatic approximation techniques, tensor factorization [2] and representation quantization [1] to generate compact models as well.

B Dominant classes

In the WEG algorithm, an important component is to decide the dominant classes from a sliding window (function DOMCLASSES in Algorithm 1). The decision used in the algorithm is fairly simple: return the classes as dominant classes that appear at least k times in a sliding window w of classification result history from the oracle h^*, where k is the minimum support number.

Here we use a simple model to analyze how to choose the minimum support number k for different window sizes w in the WEG algorithm. Suppose N is the number of total classes classified by the oracle h^* and the accuracy of h^* is a^*. If the classification result from the oracle is wrong, the probability that the oracle classifies the input to each of the other classes is assumed to be equivalent. Suppose the input sequences are drawn independently from a skewed distribution T which has n dominant classes with skew p. Denote the set of dominant classes as $\mathcal{D}$ and set of non-dominant set as $\mathcal{O}$. Based on these definitions, we can compute the probability of a single class that the oracle classifier h^* outputs given the input sequences. Consider one dominant class $\ell \in \mathcal{D}$, the probability that it is output by the oracle is:

$$\text{prob}(\ell \in \mathcal{D}) = \frac{1}{n} \cdot \hat{p} = \frac{1}{n}\left(p \cdot a^* + p(1-a^*)\frac{n-1}{N-1} + (1-p)(1-a^*)\frac{n}{N-1}\right) \tag{1}$$

And the probability of a non-dominant class $\ell' \in \mathcal{O}$ that is output by the oracle is:

$$\text{prob}(\ell' \in \mathcal{O}) = \frac{1}{N-n}\left((1-p)a^* + (1-p)(1-a^*)\frac{N-n-1}{N-1} + p(1-a^*)\frac{N-n}{N-1}\right) \tag{2}$$

index	N	a^*	n	p	w	k	p_{in}	p_{out}
1	1000	0.68	5	0.9	30	2	0.896	6.52E-5
2	1000	0.68	10	0.9	30	2	0.558	6.53E-5
3	1000	0.68	10	0.9	60	2	0.891	2.64E-4
4	1000	0.68	10	0.7	60	2	0.789	4.80E-4
5	1000	0.68	10	0.7	90	2	0.933	1.08E-3
6	205	0.58	10	0.9	90	2	0.959	0.019
7	205	0.58	10	0.9	90	3	0.872	1.31E-3
8	205	0.58	0	N/A	90	3	N/A	9.29E-3

Table 2: Probability p_{in} and p_{out} under various N, a^*, n, p and w, k. In row 8, $n = 0$ and p = N/A indicates that the distribution has no skew and is uniform across all N classes.

From Equation 1 and 2, we can then derive the probability of a dominant class or a non-dominant class that is classified at least k times in the window w by using binomial distribution. Intuitively, we hope the probability of a dominant class that appear at least k times (p_{in}) in the window be as high as possible, while the probability of any non-dominant class (p_{out}) be as low as possible. We considered different combinations of N, a^*, n, p under different w and k settings, and computed p_{in} and p_{out}. Table 2 shows how p_{in} and p_{out} changes under different settings. From the table, we can tell that with an increase in the number of dominant classes and a decrease in skew, we need to increase the size of window to have a higher probability that the dominant classes can be detected in the window. However, when the window size is larger, we also need to increase the minimum support number k (Row 6 and 7) to limit the probability that a non-dominant class appears k times. In addition, when there is no skew in the distribution (Row 8), the minimum support number k is also effective at filtering out most of the non-dominant classes. In practice, we set $k = 2$ for $w < 90$ and $k = 3$ for $w \geq 90$.

References

[1] I. Hubara et al. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016. 1

[2] Y.-D. Kim et al. Compression of deep convolutional networks for fast, low power mobile applications. *ICLR*, 2016. 1

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Twenty-sixth Annual Conference on Neural Information Processing Systems (NIPS)*, 2012. 1

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. 1