The deployment of the neural network model involves many aspects: the model needs to be simplified to fit the computing hardware constraints of the chosen device; parameter quantization within the neural network (NN) model is required to use the number representation of the hardware; firmware and hardware acceleration are necessary for speeding up the model inference; and communication ports are needed for transmitting data between the cloud server and edge devices, or among edge devices. In this chapter, the first two aspects are studied in detail, while the last two hardware-related aspects are discussed in general with details deferred to future work. Section 1 gives a summary of the seismic acquisition devices commonly used in field deployments today. Detailed information about implementing convolutional neural network (CNN) models on embedded platforms is provided to explain why these devices can support an accurate, responsive, and reliable seismic monitoring system. Section 3 tries to find a way to improve the CNN model demonstrated in Chapter **??** by visualizing filter weights within the CNN-based phase identification classifier (CPIC) model. The computational burden of the CNN is alleviated by model simplification techniques presented in Section 4. The simplified model is then quantized in Section 5 with a fixed-point representation that is likely to be used on many seismic edge devices, which further reduces the memory cost.

1 Sensors for Seismic Monitoring

Seismic sensors are the devices detecting displacement, velocity, or acceleration of the ground motion resulting from fault movement or from transient oscillations as seismic waves pass under a monitoring site. Traditional monitoring systems typically provide only sparse observations as the receivers are spaced too widely or data are not continuous in time due to cost. Research-grade, high-resolution seismic sensors are expensive. In fact, those specialty instruments are usually placed in long-term installations that are suitable for strong motions from large earthquakes. One of these stations costs tens of thousands of dollars to build and equip, including sensors, on-site data acquisition systems, telecommunications, and back-up power. However, critical seismic monitoring systems, such as earthquake early warning (EEW), that aim to detect and locate earthquakes in real-time will require long-term continuous monitoring with a densely populated array for better spatial resolution. Even the densest seismic networks, e.g., the Southern California Seismic Network (SCSN), typically do not have more than one sensor every $\approx 20\,\text{km}$, because these can cost hundreds of thousands of dollars, or more, each year to operate and maintain. The development of seismic monitoring systems in the last decade has been a pursuit of more densely observed earthquake ground motions, which has inspired improvements in both monitoring devices [**etienne2016simultaneous**] and processing algorithms [**inbal2015imaging**, **riahi2015seismic**].

Device and communication advances are opening up opportunities for very dense 1-D seismic sensor ar-

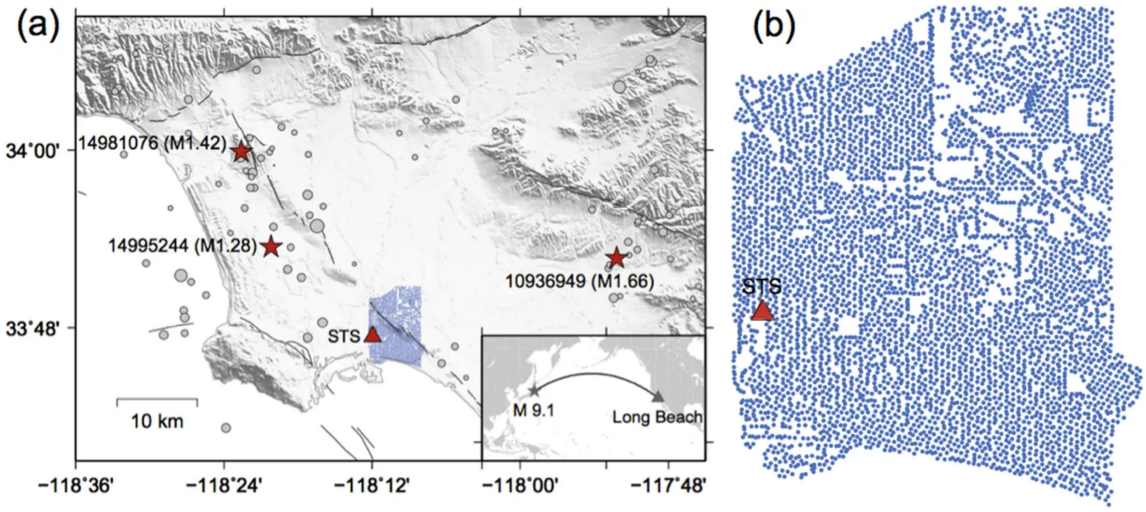


Figure 1: (a) Map of study region in **li2018high** for the Long Beach nodal array and local seismicity. (a) Blue dots are the 5200-sensor nodal array. A red triangle marks the broadband station STS belonging to Southern California Seismic Network (SCSN). Black curves denote the surface trace of mapped faults. Gray solid circles are seismicity listed in the SCSN catalog between January and June 2011, whose sizes are proportional to the magnitude. Red stars mark three selected cataloged events used for tests. The inset map shows locations of the M_w 9.1 Tohoku-Oki earthquake and its ray path to Long Beach. (b) Zoomed-in plot of the Long Beach array and the STS station.

rays with lower per-station costs. The oil and gas industry, which uses seismic sensors to conduct surveys of subsurface structures, has converted cabled sensor systems into wireless *nodal* instruments [**slater2012california**]. The nodal system eliminates the need for transport and installation of the heavy cables that once connected sensors together. Thus, these systems, which have batteries that last 30 or more days, are becoming attractive for use in earthquake studies. A network of 904 nodal nodes was deployed in a recent experiment on Mount St. Helens for two weeks and detected an order of magnitude more earthquakes under the active volcano compared to the traditional seismic network [**hansen2015automated**]. Weaker events that are unknown were detected using a six-month deployment of 5400 sensors over a ≈ 100 km area in Long Beach, California [**li2018high**] (Figure 1). In Oklahoma, a nodal array is being used to track small earthquakes moving along previously undetected fault structures that are induced by wastewater injection during the horizontal drilling process [**sweet2018community**, **dougherty2017large**]. While nodal array systems enable the deployment of dense sensor arrays at lower costs than ever before, their recording time is limited to relatively short time periods (weeks to months) on a stand-alone battery.

Systems monitoring seismic activity over mobile devices, such as a smartphone, were developed for EEW purpose [**kong2016myshake**]. These low-cost, "personal seismometers" provide opportunities for long-term installations for wider coverage, especially in cities where it is difficult to deploy seismic sensors traditionally. These systems build on the availability of inexpensive low-power embedded processors with commercial-

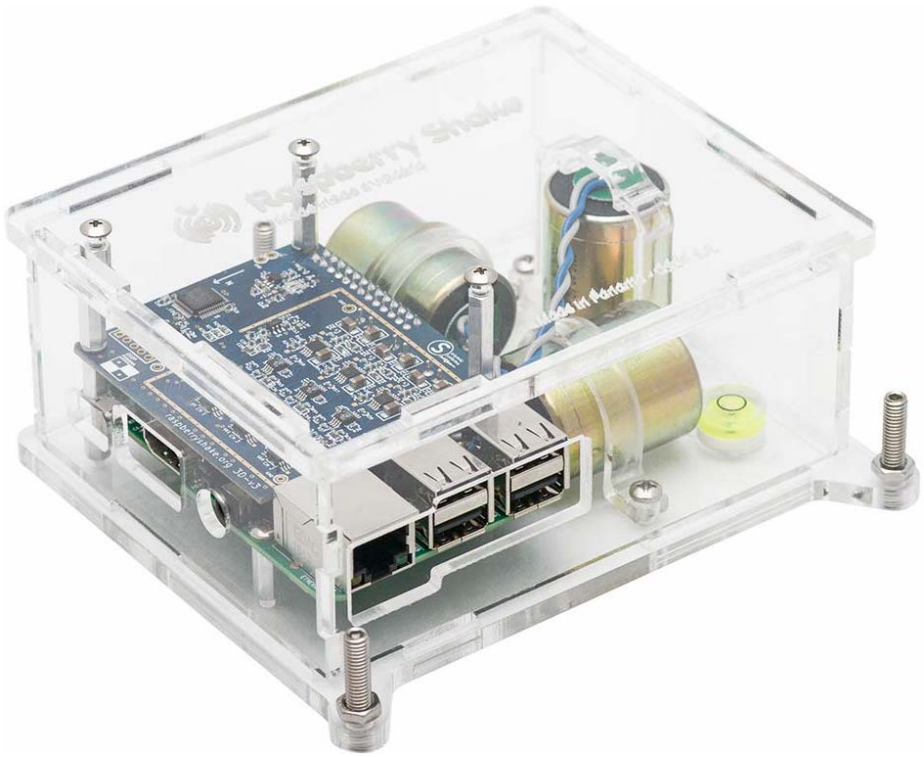


Figure 2: Raspberry Shake 3D consisting of Raspberry Pi computer plus three vertical geophones with 100 Hz sampling rate.

grade accelerometers as their seismic sensors. Accelerometers are ubiquitous in modern electronics, such as activating car airbags, rotating smartphone screens, and controlling gaming systems. Even though the sensors in these devices are of lower quality compared to scientific-grade instruments, they are adequate to capture moderate to strong ground motions and are readily available for broader coverage of large monitoring regions. Similar sensors have been installed in buildings to study the structural sway when a passing earthquake wave [**kurata2017development**].

In the wake of embedded processing requirements of real-time seismic sensors, the Raspberry Shake (Figure 2) devices were developed with a large variety of configurations containing typical seismic sensors. The onboard Raspberry Pi processor, which provides a four-core 1.4 GHz CPU and 1 GB of on-chip RAM, is a widely used embedded processing platform with extendibility potential to different types of sensor and communication modules [**upton2014raspberry**]. Raspberry Shake sensors have been added to existing local seismic networks to complement the observations for moderate to large earthquakes, including in countries with more limited seismic monitoring infrastructure [**christensen2017raspberry**]. **anthony2018low** confirmed the effectiveness of the Raspberry Shake system on both field and laboratory observations. Different studies are being conducted to validate applications of the Raspberry Shake in many scenarios, including rockfall activity monitoring [**manconi2018evaluation**], as well as local [**christensen2017raspberry**],

regional [**pulli2018seismic**], and national seismic monitoring [**calais2019can**]. The Raspberry Shake has proven to be not only a good development platform but also a viable deployment system for real-time acquisition and processing of seismic waveforms. However, due to the limited computing resource on most of the embedded processors, a pre-trained full NN model from a workstation cannot be directly deployed without further simplification.

2 Computation on Hardware

2.1 Convolution Strategies

The basis of a CNN model is the convolution operator between the input feature maps and the filter weights. Three different strategies have been proposed to implement a convolution operator, namely direct convolution, unrolling based convolution, and fast Fourier transformation (FFT) based convolution. The direct convolution computes the convolution via a sliding window of filter weights over the feature maps which computes their dot product. With proper padding and striding, the output can be formed into the desired shape. This approach is adopted by cuda-convnet2 [**krizhevsky2014one**], and Theano-legacy [**bergstra2010theano**, **bastien2012theano**]. However, this direct approach usually results in a rather implementation. **chetlur2014cudnn** used a memory heavy, but more efficient way to compute the convolution operator. Each sliding window is unrolled into a row of a larger feature map matrix and the corresponding filter kernel is unrolled into a column of filter matrix. The convolution operation is then reduced to multiplying the filter matrix by the feature map matrix. Obviously, there are redundant entries in both matrices meaning that it uses more memory to store the intermediate values than the direct approach. On the other hand, the convolution operation involves moving a considerable amount of data in and out the processor which is usually the bottleneck instead of the actual computation. Converting the convolution to matrix multiplication gives a faster run time, because it performs a higher ratio of operations per byte of data transferred. This ratio increases as the matrices get larger, so we want to form the largest matrix multiplication that the memory supports. This unrolling based convolution approach is also adopted by many popular Frameworks including Caffe [**jia2014caffe**], Torch [**collobert2011torch7**], Theano [**bastien2012theano**] and later inherited by TensorFlow [**abadi2016tensorflow**] and PyTorch [**paszke2017automatic**]. **vasilache2014fast** proposed the FFT based scheme that converts the time-domain convolution into a product in the Fourier domain. This conversion significantly reduces the computational complexity; however, the overhead of converting between the time domain and Fourier domain makes it less efficient for smaller filters. A detailed comparison of these three strategies is given by **li2016performance**. Since the filter size in the CPIC model is either 3 or 5, the

unrolling approach is preferred in this work.

2.2 Matrix-matrix multiplication

The computation for fully-connected layers (FC layers) can be easily represented as a matrix-matrix multiplication operation. Since we adopt the unrolling approach in Section 2.1, the convolution operations for the convolutional layers (CONV layers), which can be represented as a series of convolutional matrices, also consists of many matrix-matrix multiplications.

Indeed, consider a one layer in a CNN of the CPIC model as an example. There are M channels of inputs $x_m[n]$ for $m = 1, 2, \ldots, M$, each of length N, i.e., $n = 0, 1, \ldots N-1$. The CONV layer produces K channels of outputs $y_k[n]$ for $k = 1, 2, \ldots, K$, also of length N (prior to max pooling). Each input is connected to all outputs via 1-D convolution, where the filter length is L and the filter coefficients (called the impulse response in DSP) are $h_{k,m}[n]$ for $n = 0, 1, \ldots, L-1$. Thus we have MK filters – all with different filter coefficients. Altogether, there are MLK parameters needed to define all the filters. The k-th output channel, which is the sum of M filtered inputs, becomes

$$y_k[n] = \sum_{m=1}^{M} h_{k,m} * x_m = \sum_{m=1}^{M} \sum_{\ell=0}^{L-1} h_{k,m}[\ell]\, x_m[n-\ell] \qquad \text{for } n = 0, 1, \ldots, N-1 \tag{1}$$

where $*$ denotes 1-D convolution.

One-dimensional convolution for one output channel can be expressed as a matrix-vector multiplication. Since the length of the filters is much less than the input signal lengths, it is best to create the convolution matrices from the M input channels. The result is an $(N + L - 1 \times L)$ matrix $\boldsymbol{X}_m^{\text{Full}}$, where each column of $\boldsymbol{X}_m^{\text{Full}}$ contains all the signal values in the m-th input channel. Successive columns are staggered by the stride length. When the stride is one, the result is a Toeplitz matrix. Here is an example for $N = 7$ and $L = 3$ (the

subscript m is dropped from x_m), where the matrix is $(N + L - 1) \times L = 9 \times 3$:

$$\boldsymbol{X}_m^{\text{Full}} = \begin{bmatrix} x[0] & 0 & 0 \\ x[1] & x[0] & 0 \\ x[2] & x[1] & x[0] \\ x[3] & x[2] & x[1] \\ x[4] & x[3] & x[2] \\ x[5] & x[4] & x[3] \\ x[6] & x[5] & x[4] \\ 0 & x[6] & x[5] \\ 0 & 0 & x[6] \end{bmatrix} \qquad \text{Full Convolution matrix is } 9 \times 3$$

If we want the output channels to have the same length as the input channels, we must remove two rows — conventionally the first and last, or the last two, and then we call the resulting matrix $\boldsymbol{X}_m$. To get the matrix-vector form of convolution, we now define a length-N column vector for the k^{th} channel of output, $y_k[n]$, as

$$\boldsymbol{y}_k = \begin{bmatrix} y_k[0] & y_k[1] & \dots & y_k[N-1] \end{bmatrix}^T$$

and a length-L column vector for a filter between m^{th} input channel and k^{th} output channel

$$\boldsymbol{h}_{k,m} = \begin{bmatrix} h_{k,m}[0] & h_{k,m}[1] & \dots & h_{k,m}[L-1] \end{bmatrix}^T$$

The matrix-vector form for the convolution in (1) becomes

$$\boldsymbol{y}_k = \sum_{m=1}^{M} \boldsymbol{X}_m \boldsymbol{h}_{k,m} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 & \dots & \boldsymbol{X}_M \end{bmatrix} \underbrace{\begin{bmatrix} \boldsymbol{h}_{k,1} \\ \boldsymbol{h}_{k,2} \\ \vdots \\ \boldsymbol{h}_{k,M} \end{bmatrix}}_{\text{kernel}}$$

Now we put all convolution matrices into one matrix $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 & \dots & \boldsymbol{X}_M \end{bmatrix}$ and define the *kernel* for the k^{th} output as a collection of filters, $\boldsymbol{h}_k = \begin{bmatrix} \boldsymbol{h}_{k,1}^T & \boldsymbol{h}_{k,2}^T & \dots & \boldsymbol{h}_{k,M}^T \end{bmatrix}^T$. Finally, we write one matrix

product for all the outputs of this single CONV layer as

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_K \end{bmatrix} = \boldsymbol{X} \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \cdots & \boldsymbol{h}_K \end{bmatrix} = \boldsymbol{X}\boldsymbol{H} \tag{2}$$

where $\boldsymbol{Y}$ is $N \times K$, $\boldsymbol{X}$ is $N \times ML$, and $\boldsymbol{H}$ is $LM \times K$.

Figure 3 gives an example of such an unrolled convolution between a 3×6 input feature map (X_{ij}) and two three-channel convolution kernels (H_{ij} and G_{ij}). Convolution matrices of the feature map with full zero-padding are constructed for each input channel (represented in three different colors) and concatenated horizontally into one **input matrix**. Since the stride value in this example is 1, notice that the matrix structure inside each colored block of the **input matri**x is Toeplitz. If a stride value larger than 1 is used, some of input matrix rows are removed for the corresponding windows skipped due to the larger stride value. On the right hand side, the filter weights of different input channels are concatenated vertically into columns of the **kernel matrix** according to the channel order in the input matrix (using the same color code). Different filters are stacked horizontally to so that their outputs can be computed simultaneously. Note that the kernel matrix is invariant to the specific stride choice. Thus, the computation of the output (yellow) for this CONV layer can be represented as a multiplication between the input and kernel matrices.

As a result, accelerating the computation of matrix-matrix multiplication (e.g., $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$) can significantly improve the computation of the overall CPIC model, which relies on CONV layers and one FC layer. Such operations are conventionally performed by a function called general matrix multiply (GEMM). Most GEMM functions from the basic linear algebra subprograms (BLAS) are optimized for scientific computing where matrices might be as large as thousands of double-precision floating-point elements. Thus, these GEMM functions typically repack two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ into smaller shapes as shown in Figure 4 when the overall matrix size exceeds the L1 cache size of the microkernels. Each pair of these smaller matrices is fed into a microkernel to compute their product and accumulated outside for the final results. The bottom line is using fewer filters in each kernel, or fewer kernels, can significantly reduce the overall computation cost.

Another issue is number representation. When fixed-point numbers are used for the seismic acquisition system, it is natural to use fixed-point numbers for the CNN computation as well. However, when using the same GEMM functions, accumulation of fixed-point numbers outside the microkernel might be necessary but is very undesirable. Instead, using lower-precision fixed-point numbers is preferred so that the entire dot product can be computed at once inside a single microkernel at high precision as shown in Figure 4. Then the dot product result is shifted back into a compatible lower-precision fixed-point number as the output of the microkernel. This puts an additional constraint on the system design that the memory required for each dot product block needs to be less than the L1 cache size of the processor. Note that most embedded processors,

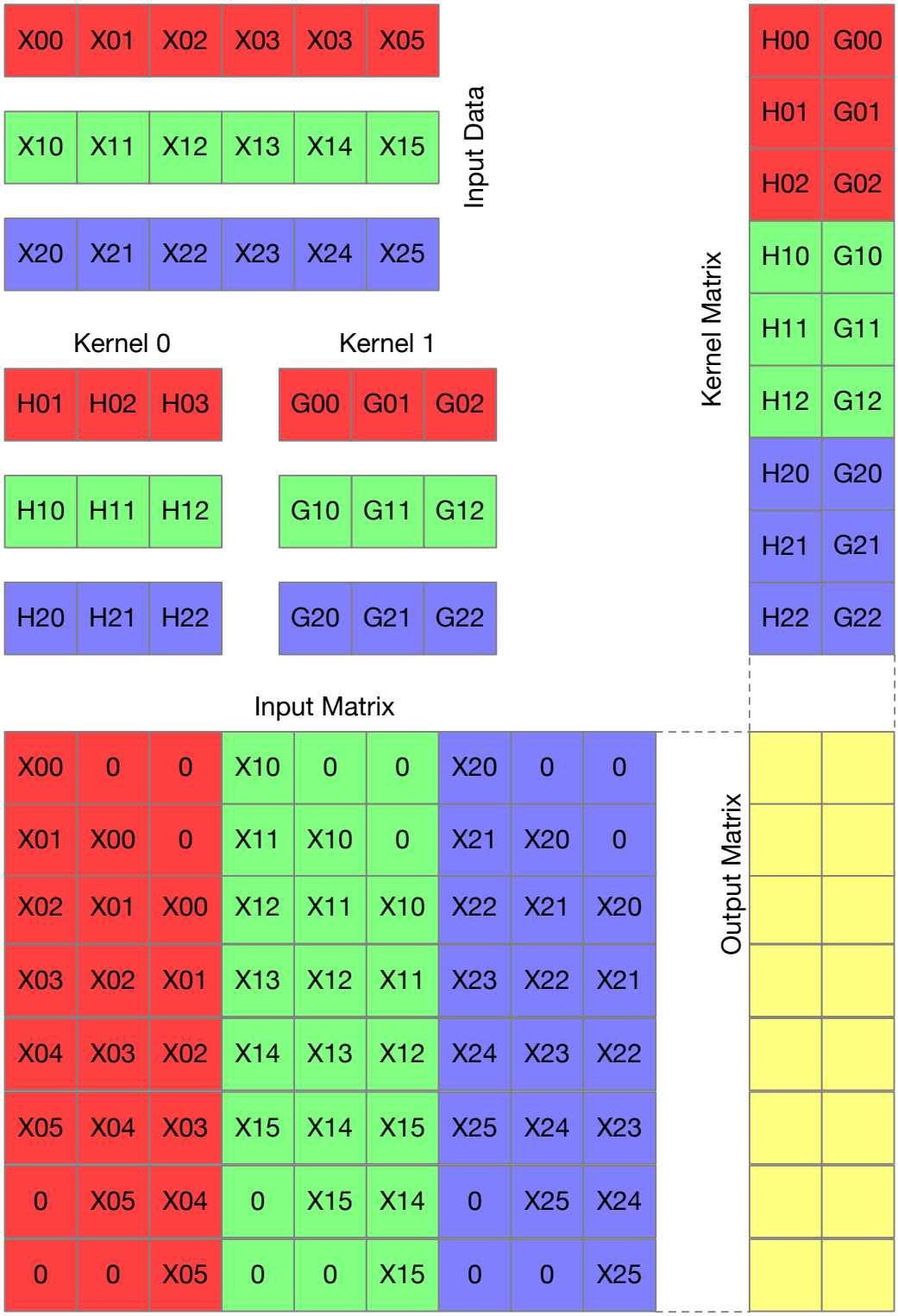


Figure 3: Convolution example (with zero-padding) between a three input channels and two size-9 kernels. Notice that $M = 3$, $N = 6$, $L = 3$, and $K = 2$ as defined in (1).

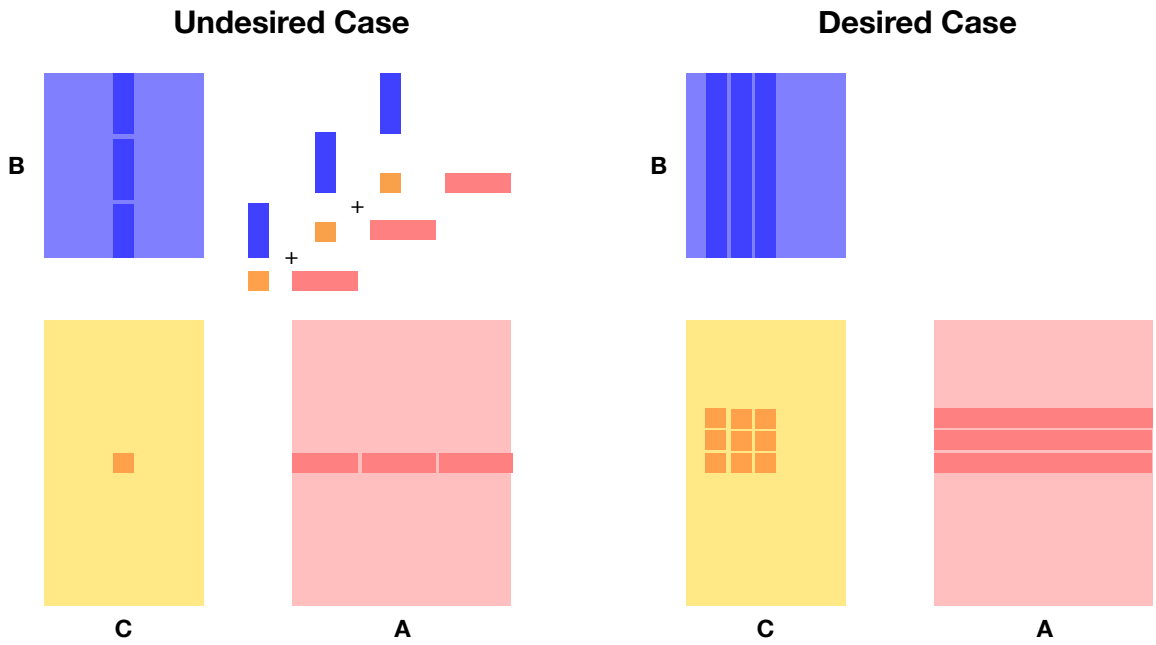


Figure 4: GEMM implementations for (undesired case, left) high-precision floating-point system where rows and columns must be broken up to compute one output of the large matrix, and (desired case, right) low-precision fixed-point system where entire rows and columns are available in the L1 cache to create small submatrices of the output.

including the Raspberry Pi, have more than 16 KB of L1 cache. This allows multiple outputs to be computed inside each microkernel as shown in Figure 4 using lower-precision numbers, which further improves the computation efficiency.

3 Visualizing CNN filter weights

To simplify the CPIC model, two approaches are studied in this section. First, by visualizing the filter weights, redundant filters can be identified and removed. After such pruning the entire CNN model must be fine-tuned and revalidated to make sure that the desired classification accuracy is maintained on the simplified model. Second, the distribution of filter weights on each CONV layer spans a relatively consistent dynamic range. This property allows us to use a lower-precision fixed-point number representation, because the small errors in number representation should not change the classification outputs very much. Both of these phenomena can be exploited to further simplify the CNN model complexity, as well as reduce the memory cost.

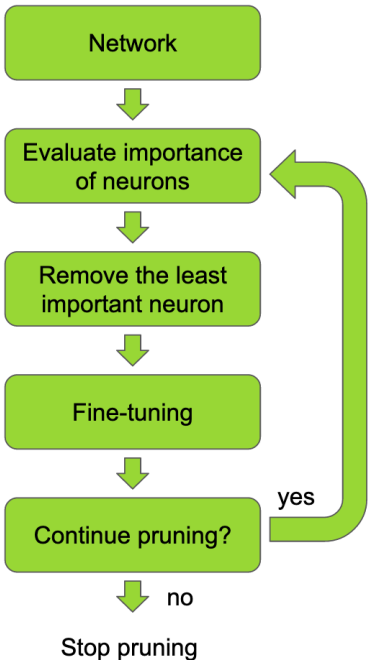


Figure 5: Flow diagram of neural network pruning as a backward filter presented as Figure 1 in **molchanov2016pruning**.

3.1 CNN Filter Weights Close to Zero

Although there are many filters within each layer of the original CPIC model, there exist some filters most of whose weights are very close to zero. These filters can be found by visualizing the magnitude of each kernel's filter weights as a 2-D image. Eliminating such kernels helps to reduce unnecessary computation and memory usage while having little effect on the final classification results. This process is known as network pruning.

The general steps of network pruning [**li2016pruning**] are shown in Figure 5: the importance of each neuron is evaluated by looking at the effect of its filter weights. A computationally costly scheme of ranking the importance of each neuron has been proposed by **molchanov2016pruning** from Nvidia; a similar approach was proposed by **anwar2017structured** as well. These techniques prune the network by running the following optimization program:

$$\min_{\mathcal{W}'} |\mathcal{C}(\mathcal{D}|\mathcal{W}') - \mathcal{C}(\mathcal{D}|\mathcal{W})|\,, \text{ s.t. } \|\mathcal{W}'\|_0 \leq B \tag{3}$$

where $\mathcal{W}$ is the filter weights, $\mathcal{C}$ is the final classification accuracy, $\mathcal{D}$ is the training data, and B is a positive integer. Although (3) is the ultimate goal one would want for network pruning, solving it requires exhaustively trying out all possible combinations of removing different redundant weights. Such a solver requires a massive amount of computing power that is not commonly available except at those organizations with

Table 1: Number of kernels in each layer of the CPIC model during every iteration of the network pruning process.

Layer	1	2	3	4	5	6	7	8	9	10	11	fc
Original number of kernels	16	32	64	64	64	64	64	64	64	64	64	64
# close-to-zero kernels	9	13	25	26	28	22	30	30	30	31	44	24
1st iteration number of kernels	8	16	32	32	32	32	32	32	32	32	16	32
2nd iteration number of kernels	8	16	32	32	32	16	16	16	16	16	0	32
3rd iteration number of kernels	8	16	32	32	32	16	16	8	8	8	0	32
4th iteration number of kernels	8	16	16	32	16	8	8	4	4	0	0	16
5th iteration number of kernels	8	16	16	32	32	16	16	8	8	0	0	32

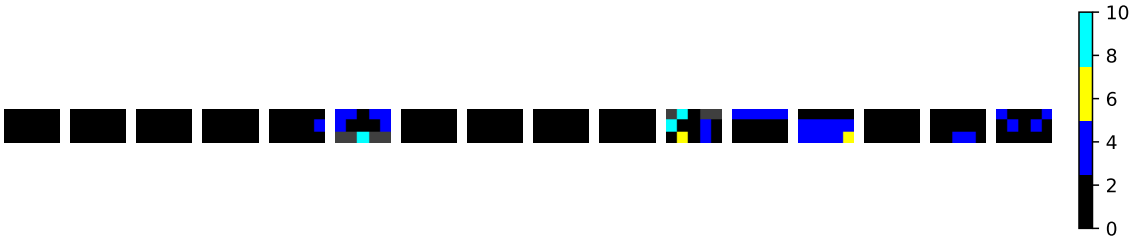
large computing clusters having thousands of graphics processing units (GPUs), such as Nvidia. More importantly, the pruned network is optimized for a fixed set of data and fine-tuning of the pruned model would require repeated runs of the optimization program in (3). This is not possible on any existing edge device, nor preferable, since the model on the edge needs to be self-adaptive to new incoming waveforms at a reasonable computing cost.

Instead of looking at individual filter weights, each filter and kernel must be treated as a whole and kept or removed altogether. By visualizing filter weights and looking for kernels whose weights are mostly close to zero, the kernels with most of their weights close to zero are identified. By removing those kernels, the minimum number of kernels are determined for each layer. The absolute value of the filter weights within CPIC layers are shown in Figures 6 – 8. At the i-th layer, the M_i filters connected to one output make up the kernel for that output. Each kernel is displayed as a rectanguar image, and one row of the image is one of the filters within a kernel. All kernels within the same layer are shown side by side in Figures 6 – 8. Note that filter weights are hard-clipped at 10 to highlight the difference between small and large filter weights. Yellow and cyan regions correspond to weights with large magnitudes, while the black regions correspond to those with close to zero magnitudes. Since there are only a limited number of kernels (K_i) in each layer, the number of kernels having all weights close to zero is easy to count. In Figure 6, nine filters in layer #1 and 13 filters in layer #2 are close to zero. This is a clear indication that the CPIC model can be simplified by removing those filters with small weights.

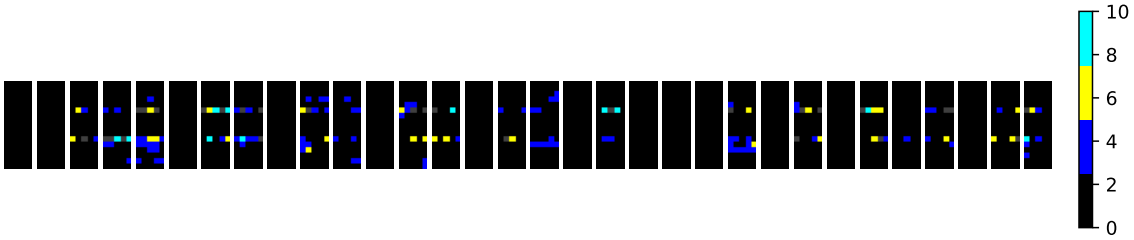
Unfortunately, the percentage of close-to-zero kernels is less on the deeper layers. Figure 7 shows the weights on layer #4, layer #7, and layer #11, respectively. Even though there are still kernels on layers #4 and #7 whose weights are almost all close to zero, many of them tend to have a small number of large weights, with the rest close to zero. On layer #11, the situation becomes similar to the first two layers where a majority of the kernels (56 out of 64) have all their weights close to zero.

In Figure 8, the weights on the FC layer are shown with dynamic ranges clipped between 0 and 1. With 64 inputs and 3 outputs, some of the weights on this FC layer are clearly close to zero. When a column of

this image is all zero, that input feature doesn't matter for classification. For Figure 8, this indicates that the final classifier made its decision on a small subset of the features in the total 64-element feature space. Thus, it is likely that the CPIC model can be further simplified by using a smaller feature space for classification.



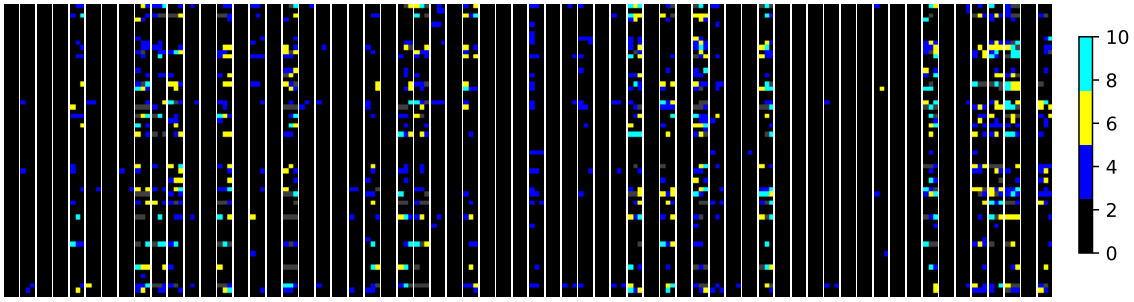
(a) Layer #1: 9 (out of 16) kernels close to zero at $k = 1, 2, 3, 4, 7, 8, 9, 10, 14$



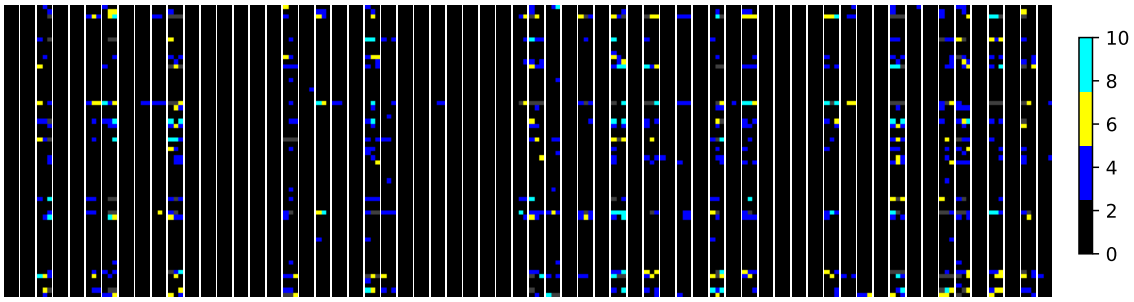
(b) Layer #2: 13 (out of 32) kernels close to zero at $k = 1, 2, 5, 8, 11, 14, 17, 19, 20, 21, 23, 25, 26, 30$

Figure 6: Absolute values of the filter weights in the first two layers of the CPIC model which use 5-point convolutions. For each subimage one row is a filter, so the horizontal dimension is $L_i = 5$, and the vertical dimension is M_i, the number of layer inputs, 3 for layer #1 and 16 for layer #2. The number of subimages is equal to the number of layer outputs, $K_1 = 16$ for layer #1 and $K_2 = 32$ for layer #2. See Figure **??** in Chapter 2 for a diagram of the entire CNN.

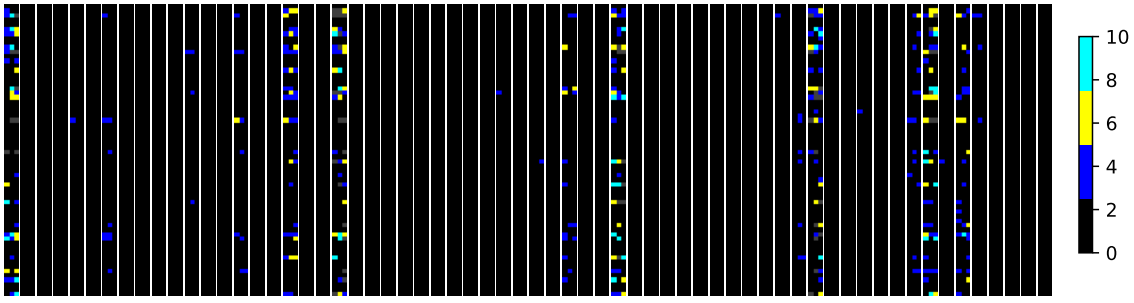
This pruning process can be conducted multiple times to further reduce the overall number of kernels (and individual filters). Empirically, the pruned model achieves better classification results by fine-tuning on a series of models whose kernels are gradually removed instead of a single model with all redundant kernels removed at once. Table 1 records the number of kernels on each layer during five pruning iterations. During the second and fourth iterations, the number of kernels on the last CONV layer is small enough that we merge it with the previous layer. This reduces the overall number of layers. After the fourth pruning iteration, the resulting model experienced a severe drop in the classification accuracy. We suspect that the model complexity was reduced too much, and the simplified model was unable to handle the variance inside the training dataset. Thus, we add back additional kernels in the fifth iteration to increase the model complexity and ultimately decided to use nine CONV layers in the final model.



(a) Layer #4: 26 (out of 64) kernels close to zero
for $k = 1, 3, 4, 7, 13, 15, 17, 21, 24, 28, 30, 31, 32, 34, 37, 40, 42, 48, 49, 50, 52, 53, 55, 56, 58, 63$



(b) Layer #7: 30 (out of 64) kernels close to zero
for $k = 1, 2, 4, 5, 8, 12, 13, 14, 15, 16, 17, 22, 25, 26, 28, 29,$
$30, 31, 35, 37, 39, 43, 47, 48, 49, 50, 54, 57, 60, 62$



(c) Layer #11: 44 (out of 64) kernels close to zero;
nonzero for $k = 1, 5, 7, 12, 15, 18, 21, 31, 33, 35, 38, 48, 49, 50, 53, 56, 57, 58, 59, 60$

Figure 7: Absolute value of the filter weights in CONV layers with length-3 filters where $L_i = 3$, $M_i = 64$, $K_i = 64$: (a) layer #4 (b) layer #7; and (c) layer #11. At least half of the kernels in most layers are close to zero. The number of kernels close to zero (black) increases, and then decreases, in the deeper layers.

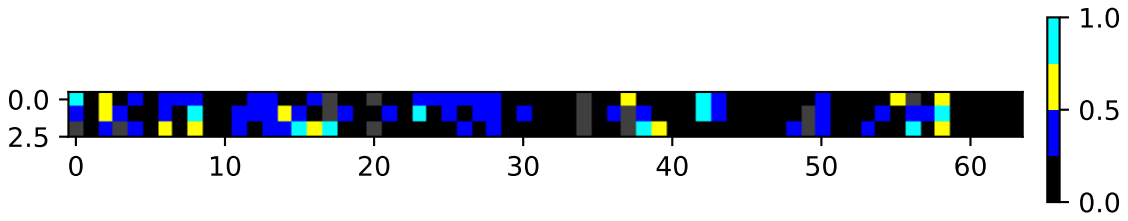


Figure 8: Filter weights in the FC layer of CPIC model with 64 inputs and 3 outputs. Each column represents the three weights from one input neuron to all three output neurons. Approximately 30 of these columns have all weights close to zero. Note that the dynamic range of these images is reduced to lie between 0 and 2.

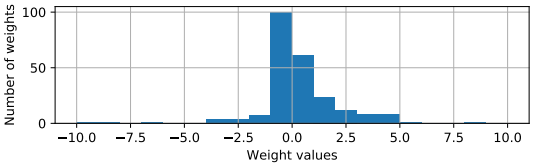
3.2 Distribution of the CNN Filter Weights

In addition to the absolute value of the filter weights, the distribution of these weights on each layer must also be considered. Distributions of the weight in all 11 CONV layers, as well as the final FC layer are shown in Figure 9 with 20 bins and a constant range from -10 to $+10$. Except for the first CONV layer, the weight distributions on all the rest of the layers resemble each other. More importantly, for all the layers these distributions are symmetric and tightly centered around zero. This phenomenon allows us to use a fixed-point number system to represent the weights in lower precision and further reduces the memory, as well as the computation cost of the CPIC model. Furthermore, a symmetric dynamic range of the fixed-point number can be adopted without a bias term due to the symmetry around zero in all the filter weight distributions. On the other hand, there is no need to consider the bias term for the feature map distribution. Since each CONV layer is followed by a rectified linear unit (ReLU) activation function, the resulting feature maps are also nonnegative which allows us to use unsigned fixed-point numbers to represent the feature map values.

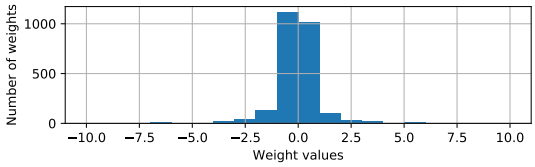
4 Model Simplification

4.1 Reducing Network Depth

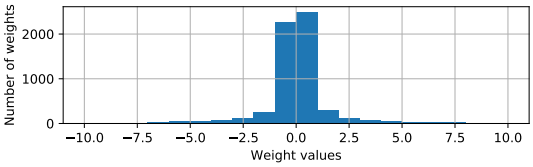
Since each CONV layer is followed by a max-pooling layer of size two, the size of the feature map in the time dimension is reduced by two following each CONV layer. Thus, the overall downsampling ratio is controlled by the number of CONV layers. In order to reduce the model complexity by removing some of the CONV layers, the duration of the input window could be shortened to have fewer time samples in the 3-C waveforms. For different window lengths T_w, the labeled phase arrival time must always be positioned at $0.25T_w$ from the beginning. As shown in Table 2, some other possible window lengths were tested, even a longer window, but a 20-s window duration works best for the Wenchuan dataset. Recall that for each manually picked phase, CPIC uses a 20-s long window starting 5 s before the pick and ending 15 s after as one window of a seismic



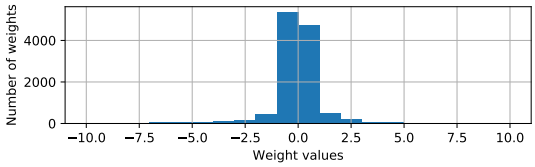
(a) CONV layer #1



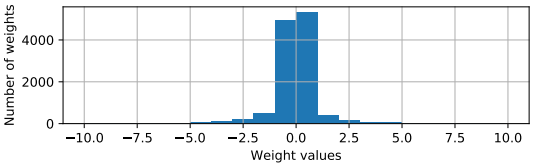
(b) CONV layer #2



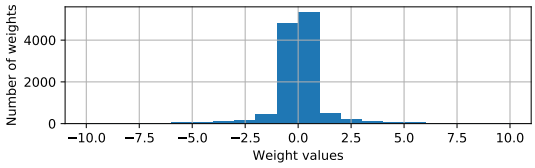
(c) CONV layer #3



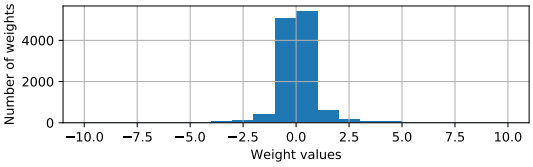
(d) CONV layer #4



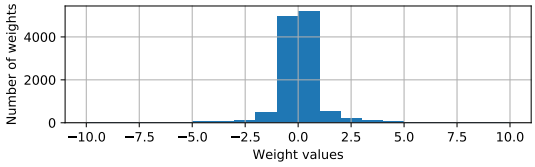
(e) CONV layer #5



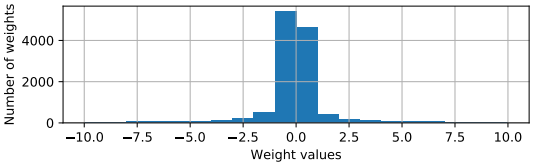
(f) CONV layer #6



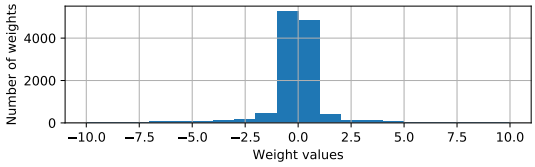
(g) CONV layer #7



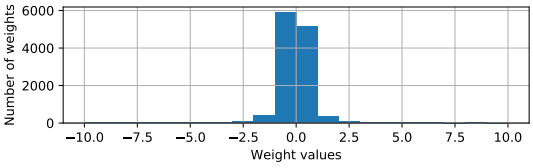
(h) CONV layer #8



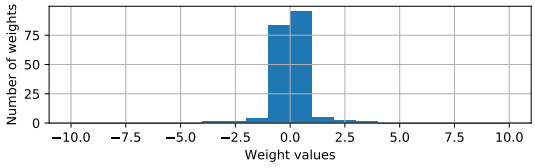
(i) CONV layer #9



(j) CONV layer #10



(k) CONV layer #11



(l) FC layer

Figure 9: Distribution of filter weights on every layer of the CPIC model before pruning. The total number of weights is $L_i M_i K_i$ for the i-th layer. Consult Table 4 for the specific values of L_i, M_i and K_i.

Table 2: Classifier accuracy (defined in (**??**)) vs. window lengths.

Window Length (sec)	2.5	5	10	20	40
Accuracy(%)	94.7	96.3	96.9	97.4	97.2

Table 3: Classification accuracy (defined in (**??**)) vs. the final feature space size.

Feature Space Size	8	16	32	64
Number of parameters	1972	7,320	30,544	107,248
Accuracy (%)	96.3	96.8	97.3	97.4

phase (Figure **??**). Thus, removing layers by shortening the input window length is not a right approach.

Another approach is possible by examining the FC layer. Since the feature space sizes in the CONV layers and the FC layer have some redundancy as shown in Figures 7 and 8, the trade-off between more CONV layers vs. a larger FC layer discussed in Section **??** can be reconsidered. In fact, the last few CONV layers perform convolutions between a very short input signal and a small kernel which is nearly the same as a fully connected structure, so it is reasonable that these could be merged into the final FC layer. Specifically, we can gradually reduce the feature space size from 64 to 4 until the max-pooling layer reduces the number of time samples from 2,000 to 8. This results in an 8×4 feature tensor which can be flattened into a 32×1 feature vector for the final FC layer.

4.2 Reducing Feature Space Size

In Section 3, it is shown that all CONV layers and the final FC layer have some redundancy in the filter weights. Thus, the overall number of filters on each layer, or the feature space size, can be reduced. To present tweaking of an individual layer and demonstrate the overall effect on the model classification accuracy, feature space sizes are reduced simultaneously with a fixed ratio on all layers. Table 3 summarizes the CNN model classification accuracy, given different levels of reduction in the feature space size on all layers. The original CPIC model has a feature space size of 64 and reaches a classification accuracy of 97.4%. By reducing the feature space size to 32, the loss in classification accuracy is negligible (only 0.1%). However, further reduction in the feature space size to 16 results in a noticeable drop in the classification accuracy, although it is still within 1% of the original model. When the feature space size is reduced to 8, a drop of a little more than 1% in classification accuracy is observed. This serves as a guideline in simplifying the CNN for improving the overall CPIC design.

Table 4: Original CNN architecture with 11 convolutional layers and one fully-connected layer (107,440 parameters).

input $(N \times M)$	kernel $(L \times K)$	max pooling	#parameters
2000×3	5×16	$2000 \rightarrow 1000$	240
1000×16	5×32	$1000 \rightarrow 500$	2560
500×32	3×64	$500 \rightarrow 250$	6144
250×64	3×64	$250 \rightarrow 128$	12288
128×64	3×64	$128 \rightarrow 64$	12288
64×64	3×64	$64 \rightarrow 32$	12288
32×64	3×64	$32 \rightarrow 16$	12288
16×64	3×64	$16 \rightarrow 8$	12288
8×64	3×64	$8 \rightarrow 4$	12288
4×64	3×64	$4 \rightarrow 2$	12288
2×64	3×64	$2 \rightarrow 1$	12288
FC-64 (64 inputs, 3 outputs)			
softmax			

4.3 Simplified Model

Combining the knowledge we gained from reducing model depth in Section 4.1 and reducing feature space size in Section 4.2, we were able to further simplify the original CPIC model as shown in Figure 10. The overall number of parameters decreases from 107,440 in the original CPIC model (Table 4) to 9,112 in the simplified model (Table 6), almost a factor of 12 reduction. Following the observation in Section 4.2, the most significant feature space size has been reduced from 64 to 32, which results in about a factor of three reduction in the number of parameters. Based on the number of close-to-zero filters in the individual layers from Section 3, the numbers of filters on the deeper layers are no longer homogeneous. Rather it decreases along with the time domain which results in a further reduction in the number of parameters. The final FC layer is also replaced by a 32 to 3 perceptron node that reduces the number of parameters by half in that layer. Table 5 shows the simplified model after the first pruning iteration as described in Section 3.1. As demonstrated in Table 1, Figure 6 shows the structure of the final simplified model after five iterations. In addition, the depth of the network is also reduced from 11 to 9 based on the observation in Section 4.1.

These simplifications in deeper layers usually result in a more significant reduction in the overall number of parameters since they usually have a larger feature space size. The CPIC model, in particular, does not need a large number of features in the deeper layers which helps to limit the number of parameters further. On the other hand, simplification in shallower layers reduces the overall computation cost while having little effect on the number of parameters. Since the data tensors in the early layers tend to have longer input length in the time domain, they require more computation to finish a convolution operation compared to a short input length. Thus, a reduction in the number of filters in the first two layers may not contribute much to the overall

Table 5: Simplified CNN architecture after first pruning iteration with 11 convolutional layers and one fully-connected layer (25,432 parameters).

input $(N \times M)$	kernel $(L \times K)$	max pooling	#parameters
2000×3	5×8	$2000 \rightarrow 1000$	120
1000×8	5×16	$1000 \rightarrow 500$	640
500×16	3×32	$500 \rightarrow 250$	1536
250×32	3×32	$250 \rightarrow 128$	3072
128×32	3×32	$128 \rightarrow 64$	3072
64×32	3×32	$64 \rightarrow 32$	3072
32×32	3×32	$32 \rightarrow 16$	3072
16×32	3×32	$16 \rightarrow 8$	3072
8×32	3×32	$8 \rightarrow 4$	3072
4×32	3×32	$4 \rightarrow 2$	3072
2×32	3×16	$2 \rightarrow 2$	1536
FC-32 (32 inputs, 3 outputs)			
softmax			

Table 6: Simplified CNN architecture after final pruning iteration with nine convolutional layers and one fully-connected layer (9,112 parameters).

input $(N \times M)$	kernel $(L \times K)$	max pooling	#parameters
2000×3	5×8	$2000 \rightarrow 1000$	120
1000×8	5×16	$1000 \rightarrow 500$	640
500×16	3×16	$500 \rightarrow 250$	768
250×16	3×32	$250 \rightarrow 127$	1536
127×32	3×32	$127 \rightarrow 64$	3072
64×32	3×16	$64 \rightarrow 32$	1536
32×16	3×16	$32 \rightarrow 16$	768
16×16	3×8	$16 \rightarrow 8$	384
8×8	3×8	$8 \rightarrow 4$	192
FC-32 (32 inputs, 3 outputs)			
softmax			

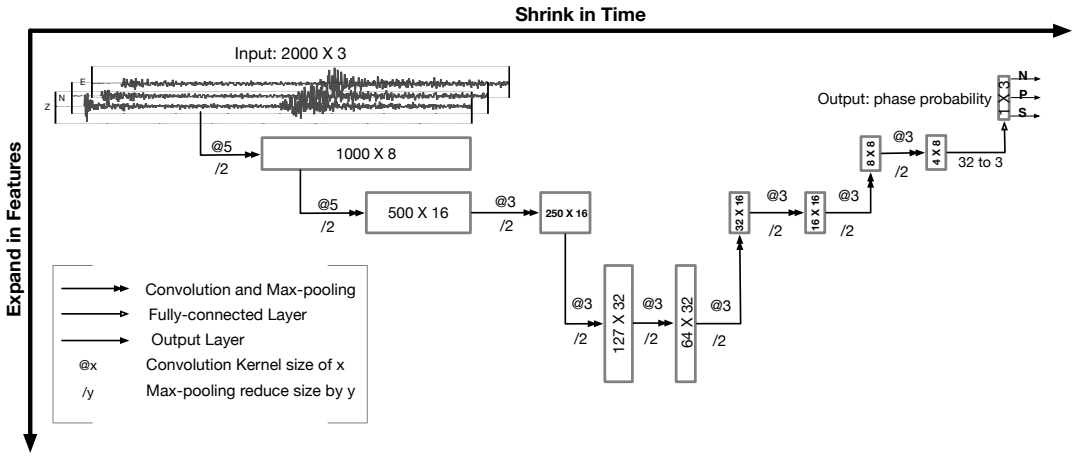


Figure 10: A diagram showing the data sizes in the simplified CNN network structure. Each input is a 3-C seismogram (20-s window) which shrinks in time but expands in the feature dimension as it passes through nine convolutional layers for feature extraction. The final layer is fully connected with three outputs that give the probabilities of a window being noise, P, and S phases.

reduction in number of parameters, but it does helps significantly in alleviating the computational burden.

The training process of the simplified CPIC model is shown in Figure 11. Comparing to the training process of the original CPIC model in Section **??**, the simplified model reaches nearly the same classification accuracy at 97.2% vs. 97.4% for the original model. However, the training process takes more epochs to converge, and the validation accuracy in the early epochs becomes quite volatile. The simplified model converges near the final stopping accuracy after more than 70 epochs, while the same happens for the original CPIC model within 40 epochs. The orange curves for training in Figure 11 resemble those in Figure 11, indicating a stable training process. However, the blue curves in Figure 11 oscillate more dramatically in Figure 11 than those in Figure **??**. The simplified model has less redundancy and can be more sensitive to perturbations in the input data when its weights have not gotten close to the optimal region. Notice that there is no over-fitting observed, as expected, on the simplified model given a training set of the same size.

5 Quantization of CNN Model

As demonstrated in Section 3.2, the effective dynamic range of the weights in each layer of the CPIC model is relatively small. Furthermore, the seismic waveforms recorded on a seismometer are usually represented in a fixed-point (integer) format using a wordlength between 12 and 24 bits. It is natural to think about quantization of the CNN model as lowering the precision of arithmetic operations, as well as reducing both the amount of computation and the memory size. Instead of using either fixed-point or floating-point numbers, a dynamic fixed-point number has been proposed specifically for quantization in neural networks

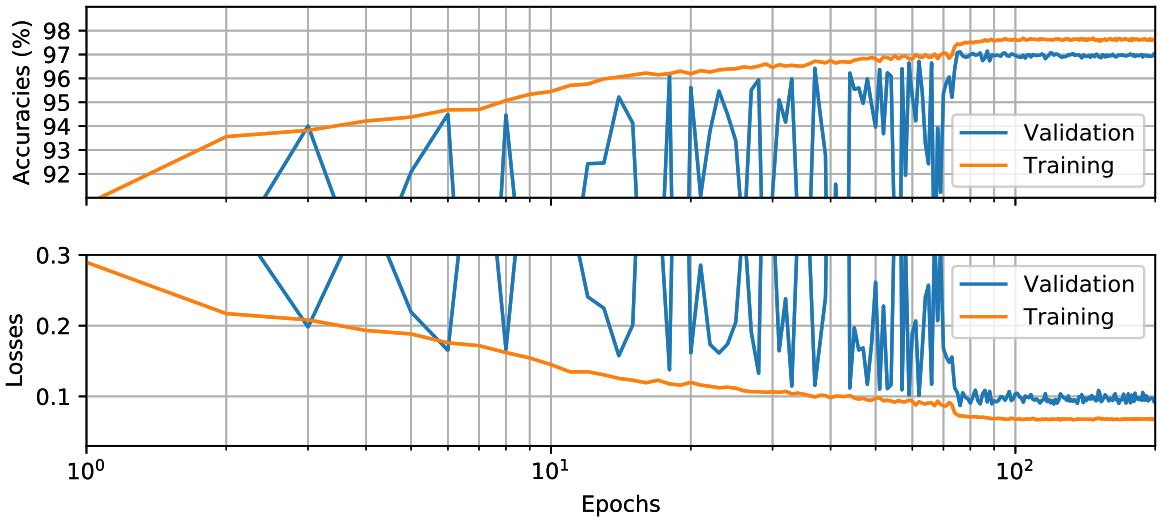


Figure 11: Training performance of simplified CPIC model on Wenchuan dataset: (a) classifier accuracy and (b) loss function against number of epochs for training and validation datasets during the CNN training process.

[**courbariaux2014training**]. Arithmetic operations are conducted in fixed-point while keeping a separate scaling factor for each neural network layer as a floating-point number. A specialized simulation scheme, which can be run on floating-point only GPUs, is designed to validate the classification accuracy of the quantized CPIC model in dynamic fixed-point numbers. Then a fine-tuning process can be used to improve the classification accuracy when the CPIC model is quantized using fewer number of bits.

5.1 Finite Word Length Scheme

Summarized in **gevers1993finite**, the performance degradations due to finite-precision arithmetic in the filter implementation and computations are:

1. quantization of the input signals into a set of discrete levels and the ensuing quantization errors
2. overflow of the computations, which occurs when the computation results are out of the range of the fixed-point hardware capacity;
3. accumulation of roundoff errors that occur at arithmetic operations;
4. quantization of the filter coefficients into a finite number of bits and the ensuing quantization errors.

The first effect, input signal quantization error, depends on the ADC used on the seismic recording device. ADC errors are usually modeled as an additive uniformly distributed white noise, which has nothing to do with the filter structure or its parameters (filter coefficients). The other three effects are influenced by the particular implementation chosen for the filter. The overflow effect is based on the wordlength capacity of

the computing system, which occurs when the dynamic range of the signals exceeds the maximum value that can be represent with a given wordlength. A proper scaling of the signals, which is a function of the signal dynamic range, is necessary to avoid serious errors caused by overflows.

The accumulation of roundoff errors after arithmetic operations and the errors induced by finite wordlength encoding of the filter weights are usually called Finite Word Length (FWL) effects. As soon as an ideal model (with full precision) is implemented in a FWL machine (an embedded device), some performance degradation due to FWL errors is inevitable. There are two basic ways of representing real numbers: fixed-point representation and floating-point representation. Internally, most computing system uses a binary number system. In *fixed-point binary representation*, the "two's complement" number representation is the most frequently used format for signed numbers. A real number x is represented in a quantized form as

$$Q[x] = K\left(-x_0 + \sum_{i=1}^{B} x_i 2^{-i}\right) \tag{4}$$

where K is an arbitrary scale factor and the x_i are binary digits which are either 0 or 1. Note that the first bit, x_0, controls the sign of x and is also known as the *sign bit*. If $x_0 = 0$, $0 \leq x \leq K$ is nonnegative; if $x_0 = 1$, $-K \leq x < 0$ is negative. The number inside the parentheses of (4) is between -1 and $+1$. Thus, any number with magnitude less than K can be uniquely represented by (4). When B is infinity, the representation is perfect; otherwise, when B is finite, we encounter a FWL effect.

The quantized representation is in the range of $-K \leq Q[x] \leq K - q$, where q is the smallest difference between any two of the total 2^{B+1} numbers, known as the *quantization step size*. Notice that the quantization error $|x - Q(x)| \leq q$ for any x, and is uniformly distributed between $-q/2$ and $+q/2$ along the x-axis regardless of the magnitude of x. With proper choice of the scaling factor K in (4), the dynamic range of $Q[x]$ can be made identical to that of a floating-point number with the same number of bits. A fundamental difference is that the quantization error of fixed-point numbers is identical all through the dynamic range, while that of floating-point numbers is small for small numbers and large for large numbers. When the relative error is more relevant than the absolute one, which is often the case, floating-point representations typically deliver a better range – precision tradeoff than fixed-point ones. However, arithmetic operations with floating-point numbers are more complicated, so floating-point coprocessors are often added in modern day computers when computation speed is an essential factor.

The dataflow in the FC layer and the CONV layers consists many multiply and accumulate (MAC) operations as demonstrated by Figure 12. The feature maps are multiplied with the filter weights, and their results are accumulated to form the outputs. For example, a feature map of m bits and filter weights of n bits are multiplied and accumulated using an adder tree such as in Figure 12. The multiplication results in an $m + n$

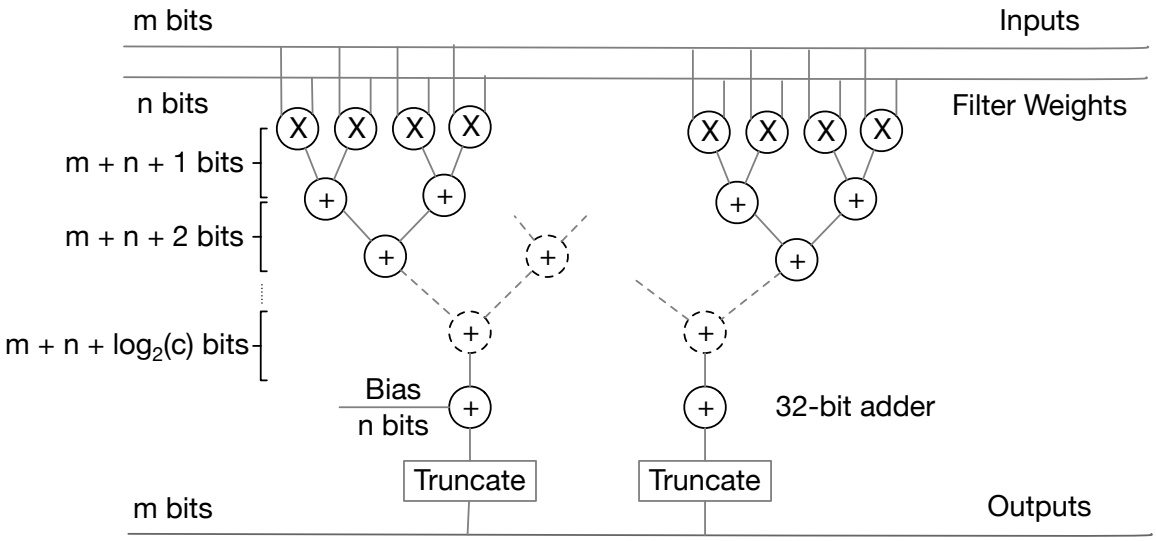


Figure 12: Fixed-point number operation in convolutional neural networks. Adapted from Figure 5.1 in **gysel2016ristretto**.

bit output product, and the additions at each level of the tree add one more bit. In the last level, the bit-width is $m + n + \log_2(c)$, where c is the number of multiplications per output channel. After filtering, a bias is added to form the output which is then truncated to the lower precision representation by the shifters. Thus, our goal in quantization design is to find a good balance between reducing the bit-widths (m and n) and maintaining a good classification accuracy for the CNN. For simplicity, m and n are chosen to be the same in this study.

5.2 Inference using the Quantized CNN Model

To accommodate the large dynamic range difference in each layer of the neural network, a *dynamic fixed-point* scheme was proposed for neural networks [**courbariaux2014training**]. It was proposed originally for faster training; however, **gysel2016hardware** later demonstrated that this dynamic fixed-point scheme has more value in approximating NN models during inference.

Similar to a block floating-point number format, the dynamic fixed-point representation uses a scaling factor S^l, which is a power of 2, represented in floating-point for scaling each NN layer. Effectively, S^l determines the location of the binary point of each number in that layer and thus controls the the fractional length of the fixed-point number. The filter weights are still represented as fixed-point numbers using this shared scaling factor as in (5):

$$x^l = S^l \left(-x_0^l + \sum_{i=1}^{B} 2^{-i} \right) \tag{5}$$

Comparing with (4), each layer has a dynamically changing scale factor S^l instead of a global fixed scaling

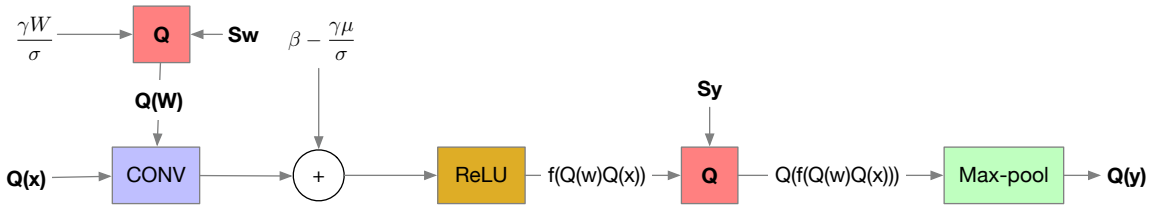


Figure 13: Flow diagram of a CONV layer quantized using the dynamic fixed-point number system. Notice that the batch norm has been absorbed into the filter weights W. There are two scaling factors: S_w for filter weights and S_y for feature maps.

factor K. However, the same S^l is shared within a single NN layer, which is different from the exponent x_e in floating-point which can vary from weight to weight. More importantly, the evaluation of this shared scaling factor can be delayed until the last layer of each branch. Since the CPIC model only has one branch, there will be only one global scaling factor in floating-point at the end. During CNN inference, the filter outputs are computed in fixed-point, where only the relative values, instead of the absolute values are tracked. Since both ReLU and max-pooling, at the layer output, operate based on comparing numbers, relative values are sufficient.

Figure 13 shows the overall inference pipeline of the quantized CNN model. The input x and filter weights W are both quantized into a B-bit representation as $Q(x)$ and $Q(W)$ respectively. The CONV layer output will be a $4B$-bit fixed-point representation $Q(W)Q(x)$, which will be bottom clipped by ReLU. The precision of the nonlinearly activated data $f(Q(W)Q(x))$ will then be reduced to B bits before passing into the max-pooling layer. Notice that the Q block can be performed after the max-pooling layer; however, it is faster to find the max value in the fixed-point representation rather than in floating-point. There are two scaling factors, S_w and S_y, involved in the process. Combining those with the scaling factors S'_w and S'_y from the previous layer, denoted as $S_x = S'_w S'_y$, the global scaling factor can be computed as

$$S = S_0 \prod_{i=1}^{N} S_w^i S_y^i \tag{6}$$

where S_0 is the input scaling factor, S_w^i and S_y^i are the dynamic scaling factors for the i^{th} layer, and N is the number of layers.

The batch norm layers are omitted in Figure 13 since they are usually absorbed into the previous CONV layer. Indeed, the CONV layer operation

$$y = Wx + b \tag{7}$$

where W is the filter weights and b is the filter bias, can be combined with the batch normalization layer (BN

layer) operation to obtain

$$y = \gamma \left(\frac{x - \mu}{\sigma} \right) + \beta \tag{8}$$

where μ is the sample mean, σ is the sample standard deviation in the current batch (a collection of training samples), γ is the scaling factor and β is the bias factor, both of which are learnable parameters in the training process. The result of combining (7) and (8) into a CONV layer operation is (9) below

$$y = \gamma \left(\frac{Wx + b - \mu}{\sigma} \right) + \beta = W'x + b' \tag{9}$$

where $W' = \frac{\gamma W}{\sigma}$ and $b' = \beta - \frac{\gamma \mu}{\sigma}$ are the filter weights and bias of the new CONV layer with the BN layer absorbed. This is also the reason that CONV layers followed by a BN layer do not need a bias in the filter coefficients.On the other hand, the BN layer dynamically estimate the sample mean and variance of each batch during the training process while using fixed values during validation. The dynamically estimated batch statistics are accumulated by an exponential moving average, i.e.,

$$S_t = \begin{cases} Y_1, & t = 1 \\ \alpha Y_t + (1 - \alpha) S_{t-1}, & t > 1 \end{cases} \tag{10}$$

where Y_t is the estimator of the current batch, S_t is the accumulated estimator of all batches, and α is the exponential decay of the weighting.

Given enough number of bits, the direct quantization of most CNN models is sufficient to approximate the original floating-point model as observed in **krishnamoorthi2018quantizing**. However, a fine-tuning process can be developed to obtain improvements over the directly quantized models.

5.3 Fine-tuning the Quantized CNN Model

Fine-tuning of a quantized CPIC model is based on a simple assumption. The optimal quantized model should be similar enough to the original floating-point model that with small tweaking of filter weights, the overall classification accuracy can be recovered. Gradient-based approaches can still be used to fine-tune the quantized weights of the neural network; however, an approximation of the derivative of the quantizer operator is needed for the chain rule to pass the gradient during the backpropagation process. The quantizer is a nonlinear and non-differentiable function as shown in Figure 14 (left), but the piecewise linear approximation in Figure 14 (right) is a function commonly used as a proxy for the quantizer's derivative. The exact outputs of the nonlinear quantizer are used in the forward pass, while the approximated derivatives are used in the backward pass. This discrepancy between forward and backward passes should be negligible if the quantizer

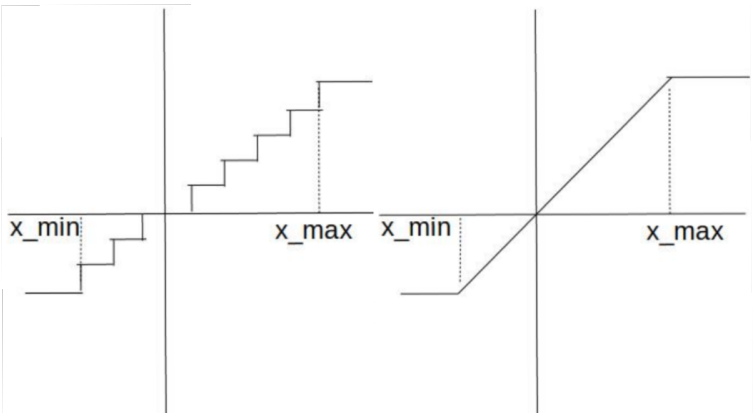


Figure 14: Simulated quantizer (left) and an approximation (right) for the purpose of calculating its derivative, from Figure 1 in **krishnamoorthi2018quantizing**.

has enough number of bits. Indeed, as the number of bits in the quantizer increases, the step size decreases and the closer two functions in Figure 14 resemble each other. When an infinite number of bits are available, these two functions are equivalent.

Different input data will result in a different scaling factor S_y^i for the i^{th} layer; however, when deploying the model, each layer needs to have a fixed scaling factor. One could look at all the known input data and use the most conservative scaling factor to avoid any possible overflow problem. However, this practice almost always results in a too conservative choice that sacrifices a significant amount of resolution for some rarely occurring extreme values. A reasonable amount of clipping is instead allowable, but it is hard to know in advance the exact amount of clipping which can be tolerated. In fact, these scaling factors on each layer can also be learned during the fine-tuning process. Similar to the sample mean and sample standard deviation in the BN layer, we use an exponential moving average to record the instantaneous scaling factor each time data pass through the quantized model. Once the fine-tuning is finished, the exponential moving average has seen the entire training set multiple times with different combinations. Each time, the exponential moving average records the most conservative choice on the current training batch with $\alpha = 0.3$. When a small portion of those training data has large magnitudes, it only affects the current batch. In the long run, the exponential moving average will smooth the overall scaling factor estimation, allowing a few overflows on some batches while being conservative for most batches. This behavior tries to find the optimal scaling factor that uses the dynamic range most effectively without explicitly validating all possible scaling factor choices on the entire training dataset. Some signal saturation due to the overflow effect is acceptable here because the output activation map of the convolutional layer will be passed into another nonlinear ReLU activation function. The existence of these nonlinearities is expected by the CNN model. Thus, the degradation of performance can be partially recovered via updating the weight values during fine-tuning process.

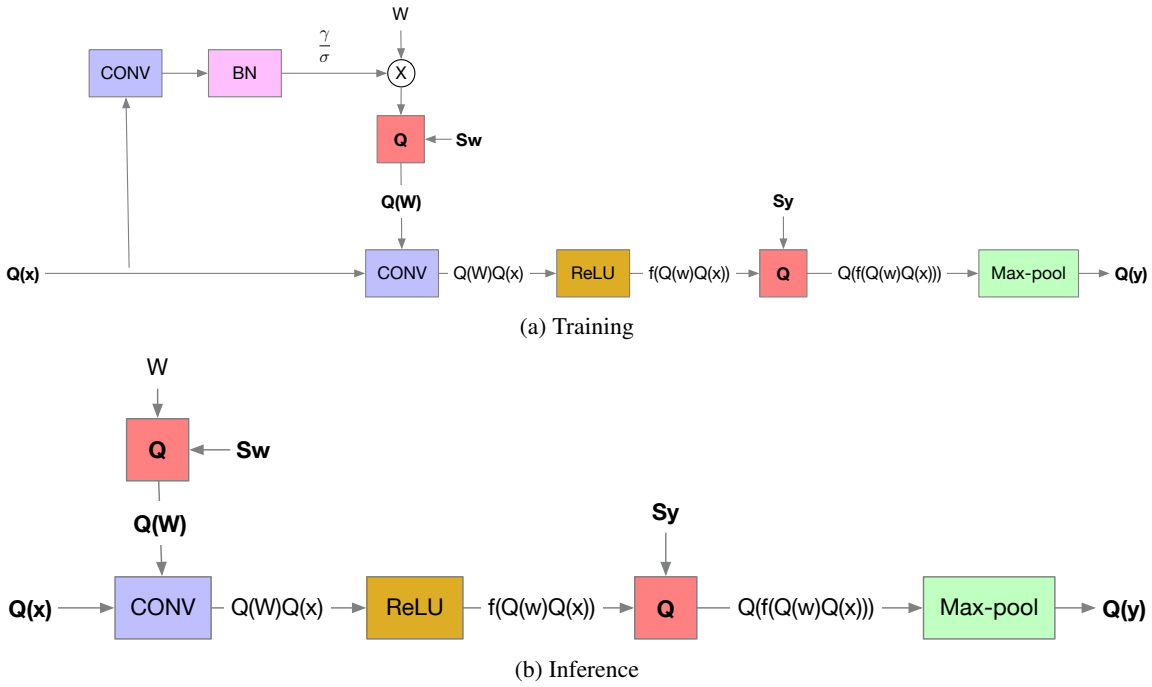


Figure 15: Fine-tuning process for quantized CNN model with the BN layer incorporated into the CONV layer.

Another challenge for fine-tuning the CPIC model, in particular, is the BN layer. Since the BN layer will be absorbed into the previous CONV layer, the quantization of the combined CONV layer needs to be simulated during the training process. Both μ and σ need to be estimated for each training batch, so the CONV layer needs to be run twice on each iteration. As shown in Figure 15a, during the training process, feature maps are passed into the CONV layer with full precision weights followed by the BN layer. This is used to estimate the sample mean μ and sample standard deviation σ as accurately as possible. The estimated μ and σ are then used to simulate the BN layer value for absorption. The new CONV layer with the BN layer absorbed is then quantized following the same procedure shown in Figure 13. During inference, the new bias b' is added back after the CONV layer whose weights include the absorbed BN layer coefficients as shown in Figure 15b.

Using the training and inference scheme shown in Figure 15, the simplified CPIC from Section 4.3 is quantized into lower precision. Figure 16 shows the evolution of the training and validation processes vs. epochs for the quantized model in 8-bit fixed-point. After an initial drop in validation accuracy, the fine-tuning process gradually recovers most of this loss of classification accuracy due to quantization in about 70 epochs. So, the fine-tuning processing was run for 100 epochs instead 200.

Table 7 summarizes results for different numbers of bits: the simplified floating-point model is quantized into dynamic fixed-point numbers using 12, 10, 8, 6, and 4 bits. The most recent high-end seismic sensors

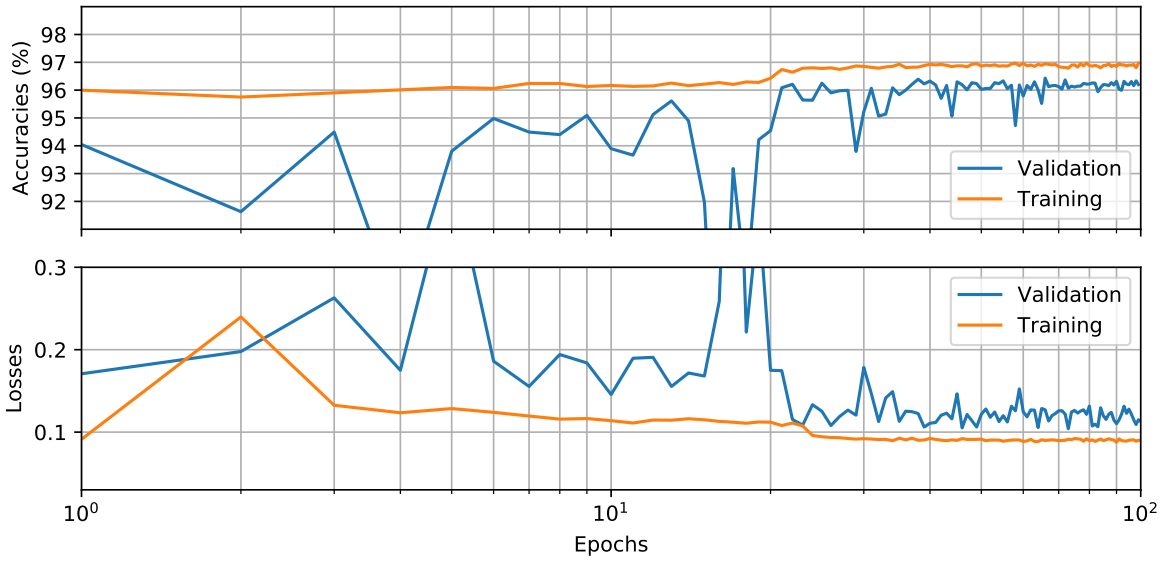


Figure 16: Training performance of simplified CPIC model quantized with 8-bit fixed-point numbers on Wenchuan dataset: (a) classifier accuracy and (b) loss function against number of epochs for training and validation datasets during the CNN training process.

record waveforms with 24-bit ADCs, while most inexpensive devices record with at least 12 bits. [TODO] say what we stopped at 12 bit. The first row of Table 7 shows the classification accuracy difference between the simplified CPIC model in floating-point and the lower-precision model with only the filter weights quantized. This is a reference point for other quantization experiments. If the assumption that the quantized model is similar to the floating-point case holds, the fine-tuned model should never have classification accuracy higher than the ones shown in the first row. This is true for all but the 4-bit case where the fine-tuned model performs significantly better than the weight-only quantized model. In this case, the weights in the fine-tuned model must be dramatically different from the original floating-point numbers making it inappropriate to use the proposed scheme to fine-tune the model for classification recovery. Indeed, the step-function-shaped 4-bit quantizer is significantly different from a 45-degree line making the approximation shown in Figure 14 invalid. Notice that the classification accuracy is better without fine-tuning in the case of 12 bits. This agrees with our assumption that the optimal quantized model is close to the floating-point model given enough number precision.

The fully quantized models are shown in the second row of Table 7. The value is slightly inflated as the exponential moving average for scaling factor estimation has been turned off for these models. The scaling factors used in validating them are dynamically computed to avoid any possible overflow problem. Since it is impossible to have a different scaling factor for different input data, the accuracy shown on the second line is slightly better than it should be. After one epoch of training, the exponential moving averages for the scaling factors have seen all training data; so the averages of the scaling factors does not change much for the model

Table 7: Validation accuracy of the quantized CPIC fixed-point model with a varying number of bits. Cases with less than 1% accuracy differences are marked in green; those with differences between 1% and 5% in orange; those with differences over 5% in red. The first row is unrealistic where filter weights are quantized but the feature maps are not, but it serves as a reference.

Number of Bits	12	10	8	6	4
Weight-only quantization (%)	97.22	97.16	97.08	95.97	73.54
Fully quantized model (%)	97.18	96.79	95.25	77.93	56.82
Fine-tuned one epoch (%)	96.95	96.76	94.66	89.46	69.89
Fine-tuned 100 epochs (%)	97.13	97.01	96.51	92.70	85.66

on the third row of Table 7. Notice that the fine-tuning process completely recovers the classification accuracy for a 12-bit model. The 10-bit and 8-bit cases can be recovered with less than 1% difference in classification accuracy. However, the 6-bit model can only be recovered within a 5% difference making it borderline for feasible usage. The 4-bit model, as discussed before, is not appropriate for the proposed fine-tuning process and has more than a 10% drop in classification accuracy even after the fine-tuning process.

Currently, the quantization process is simulated on GPUs in floating-point with artificially reduced precision. Thus it can correctly model the noise and classification error due to reduced precision; however, the simulated fine-tuning process for the quantized model is much slower than its counterpart for a floating-point model. As more and more general purpose GPUs start to support half-precision, or even quarter-precision integers, this fine-tuning may be conducted with actual fixed-point numbers, which would significantly improve its speed. Moreover, when the embedded processors start to support the acceleration of matrix-matrix multiplication of low-precision integers, we may even be able to fine-tune the CPIC model directly on the embedded processor of an edge device.

6 Deployment on Embedded Devices

The simplified CPIC model quantized in dynamic fixed-point is now ready for deployment on embedded devices. Due to the lack of compatible firmware on the Raspberry Pi board, the evaluation of this deployment of quantized models must be deferred to future work. Note that this thesis provides all the necessary tools for converting a validated floating-point model to the target fixed-point model. In the this section, the floating-point number models are benchmarked instead.

The original CPIC model and simplified CPIC models from Section 4 are tested on a four-minute segment of 3-C waveforms from the Mariana dataset. As shown in Figure 17, both P and S-wave characteristic functions from the two models have a peak near the corresponding manual pick times. Although it is counterintuitive, arrivals of P-waves are more challenging to pick on the Mariana dataset than those of S-waves.

Table 8: Benchmark of CPIC model computation times on Core i7 laptop vs. Raspberry Pi device processing a four-minute three-channel waveform from Mariana dataset.

Model	Original CPIC	Fine-tuned Original CPIC	Simplified CPIC	Fine-tuned Simplified CPIC
i7 Laptop	0.65 s	0.55 s	0.26 s	0.26 s
Raspberry Pi	66.47 s	66.68 s	34.09 s	37.68 s

This is primarily due to the source mechanism in the Mariana subduction zone which produces P-wave arrivals with gradually increasing magnitude instead of more abrupt changes from the noise floor. This is very different from cases in the Wenchuan dataset that the CPIC model was originally trained on. Thus, having the CPIC model fine-tuned on a small subset of the Mariana dataset should help with the picking results. Indeed, when fine-tuned on 2,000 samples from the Mariana dataset, the simplified CPIC model shows a significant improvement in picking accuracy for P-waves as shown in Figure 17(f). However, this is not true for the original CPIC model. When fine-tuned on 2,000 samples, the picking results hardly change for any of the predicted P-wave arrival times in Figure 17(c). We do observe some improvement in denying surface waves that come after S-waves, which were mistakenly recognized as S-waves previously without fine-tuning. After expanding the number of samples for fine-tuning to 14,000, the picking results in Figure 17(d) show some improvement for P-waves. This phenomenon may be caused by the difference in the model sizes. The original CPIC model has more than ten times the number of parameters compared to the simplified model, which means it naturally requires more training samples to reach a stable model during the fine-tuning process.

When comparing the computation time used for processing such four-minute waveform, the simplified model is consistently faster than the original CPIC model. Shown in Table 8, the time consumed by simplified model is about half of that by original CPIC model for both i7 laptop and Raspberry Pi device. This makes the simplified model more favorable. Notice that there is no GPU used for both testing cases. Computation time on Raspberry Pi is roughly 100 times of that on the i7 laptop since both the CPU clock frequency and the RAM speed on the Raspberry Pi is considerable smaller than those on the i7 laptop. However, since the computation times on both cases are significantly less than the signal duration, it is feasible to implement a real-time processing system both i7 processors as well as the Raspberry Pi devices.

Although modern hardware accelerators for floating-point arithmetic usually claim comparable efficiency to their fixed-point counterparts, the fixed-point accelerators are still cheaper to implement, consume less power, and are more ubiquitous on low-end embedded devices. Since seismic waveforms are originally recorded in fixed-point numbers, processing the data in fixed-point also has the advantage of avoiding continuously converting input waveforms from fixed-point numbers to floating-point numbers. Finally, notice that many fixed-point processors have the basic processor working at 4-bit word length. Higher precision bit computation is conducted by combining multiple basic processors. This makes selecting the number of

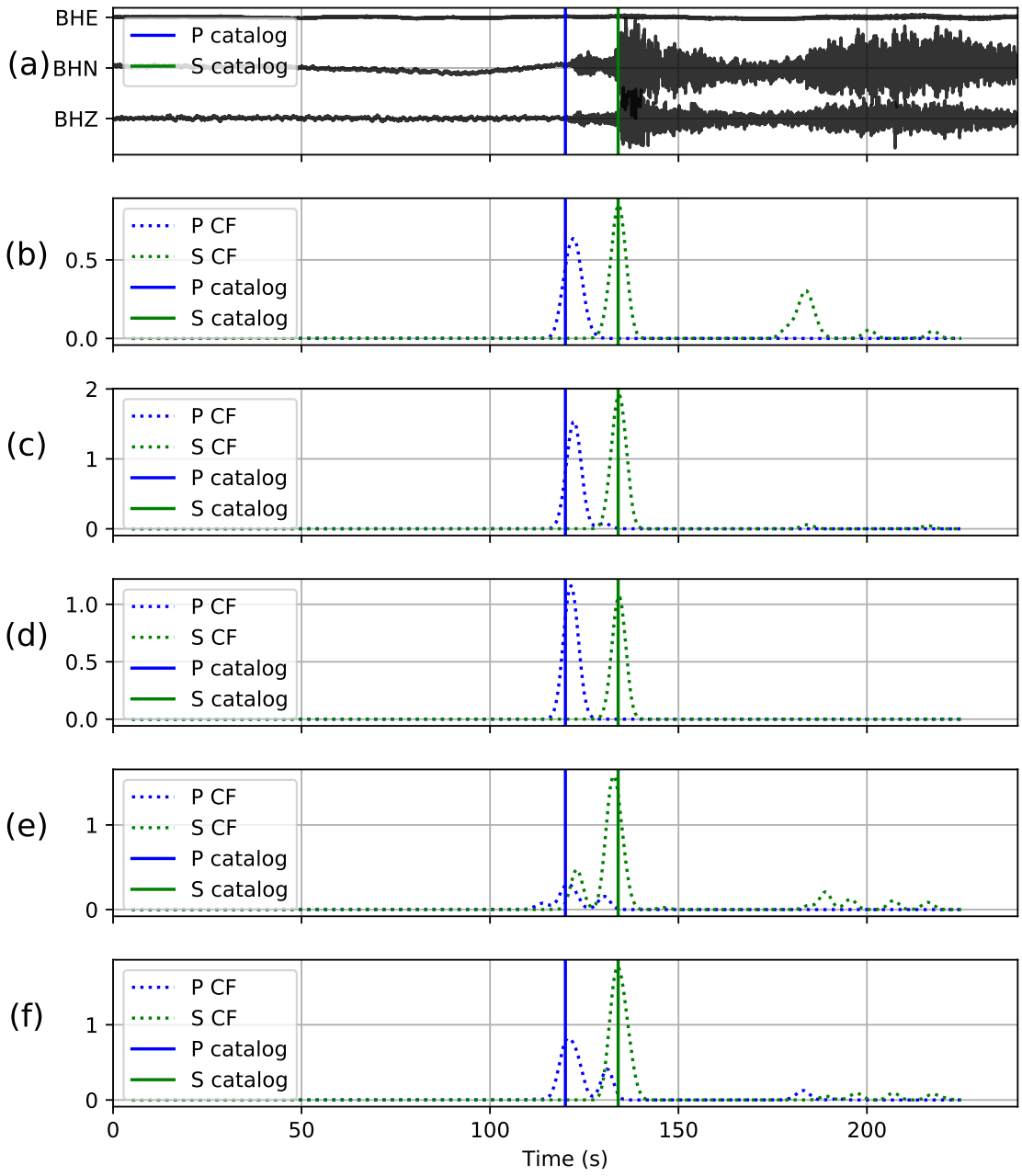


Figure 17: Picking results for five CPIC models on a four-minute 3-C waveform from Mariana dataset. The plots from top to bottom panels are (a) three-component waveform; P-wave and S-wave picking characteristic functions of (b) original CPIC model trained on Wenchuan dataset; (c) original CPIC model fine-tuned on 2,000 samples from Mariana dataset; (d) original CPIC model fine-tuned on 16,000 samples from Mariana dataset; (e) simplified CPIC model trained on Wenchuan dataset; and (f) simplified CPIC model fine-tuned on 2,000 samples from Mariana dataset. Note that the vertical blue and green lines indicate manually picked arrival times for P-wave and S-wave, respectively.

quantization bits favor using a multiple of four, e.g., 12-bit and 8-bit choices in Table 7 would be preferred.