Electronic Design

By DANIEL EHRHARDT, AI Software Engineer,
Recogni, www.recogni.com

Three Tips for Boosting CNN Inference Performance

Large performance gains with minimal changes in result are possible by optimizing convolutional neural network models.

This article presents three tips that can help you significantly improve the performance of convolutional neural network (CNN) architectures for inference tasks. These tips are based on Recogni's experience in successfully converting numerous neural networks (NNs) that have been trained on tasks ranging from simple image classification to more sophisticated challenges including semantic segmentation and 2D and 3D object detection. There's real-world experience behind these tips.

Image classification is a computer vision application that uses a NN to recognize objects in a picture or a video frame by extracting and recognizing features in the image. A NN applied to semantic image segmentation attempts to label each pixel of an image with a corresponding object class.

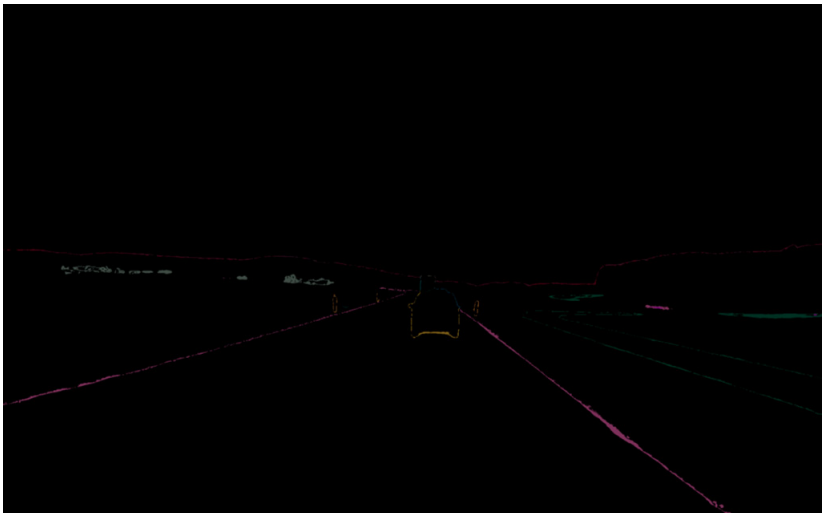
Object detection goes even further and requires even more computing power to draw bounding boxes around the objects, which means predicting their position in the image and size. It's an essential technology needed to enable autonomous vehicles (AVs), informing the vehicle about its surroundings from cars and bicycles to traffic signs and lights and road lane markings.

Training Neural Networks

Typically, NNs are trained using a cluster of servers bolstered by general-purpose graphics processing units (GPGPUs). For accuracy, NN training almost universally relies on floating-point number representations for the NN's weights and activations. Due to the use of floating-point representations and the sheer number of computations needed, NN training requires an immense amount of computing power.

If that type of compute was trivially used during inference in embedded and edge applications, it would be prohibitively power-hungry—especially for battery-powered mobile applications such as AVs. Thus, care must be taken to reframe the inference problem in ways that reduce the amount of computation needed while maintaining the required accuracy.

One of Recogni's semantic segmentation CNNs that was



1. A semantic segmentation CNN developed using the three performance tips in this article produces results that are extremely close to ground truth. Differences between the CNN's predictions and ground truth appear in color and matches appear black. The differences are minuscule.

2. The top half of the figure shows a scene overlaid with 3D bounding boxes generated by the network running at 32-bit floating-point precision. The bottom half of the figure shows the 3D bounding boxes predicted by the Recogni 3DOD pipeline after optimization of the weights and activations using the three tips described in this article. Again, the differences between the ground-truth and predicted bounding boxes are minor.

trained on the A2D2 dataset—a driving dataset of video published by Audi—achieved a mean intersection over union (IoU) score of 83.78%. It even surpassed the original paper's reference performance after having been compressed and quantized to Recogni's proprietary mathematical systems.

The IoU score specifies the amount of overlap between the predicted and ground-truth bounding boxes. It's the ratio of the intersection (the overlap) of the predicted bounding boxes and the ground-truth (actual) bounding boxes divided by the union of these two boxes.

If the prediction is perfect, the score is 100%. If the prediction is so poor that the predicted bounding boxes don't overlap the ground-truth bounding boxes at all, the score is zero. Semantic segmentation is used in AD/ADAS for various applications such as classifying the road into drivable and non-drivable segments or identifying parking spots and sidewalks.

A single numeric score can often fail to capture all of the details required to judge CNN performance and the IoU metric is no exception. *Figure 1* subtracts the pixel-value predictions of the original CNN with the predictions of a converted network for a single sample of the dataset. Differences between the CNN's predictions and ground truth appear in color and matches appear black.

Figure 1 shows that the converted CNN reproduces the original predictions almost perfectly, because almost all pixels in it are black. The differences between ground truth and prediction are minuscule.

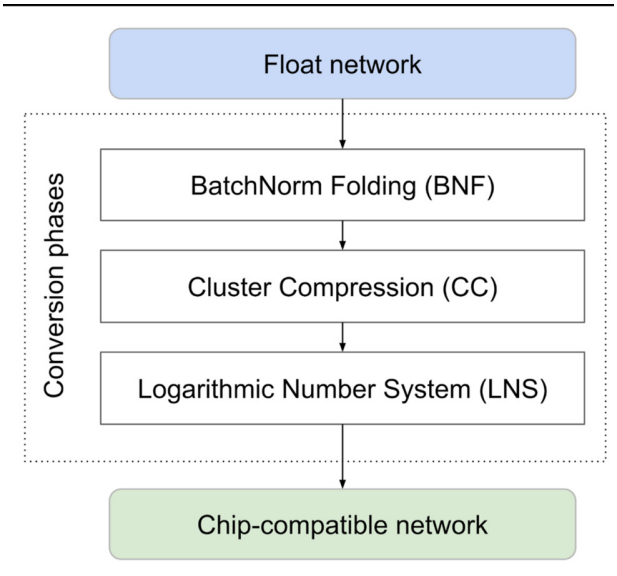
3D Object Detection

3D object detection (3DOD) as a task that AVs need to perform is among the most challenging to convert, given its highly regressional nature (meaning not just classifying objects or pixels, but gradually inferring the size, distance, or orientation of objects).

Recogni's Stereo 3DOD pipeline has a traditional deep neural network (DNN) backbone and multiple detection heads to estimate object depth from left and right image pairs and then casts the network's predictions as 3D bounding boxes located in space. These 3D bounding-box predictions are evaluated based on their position, dimensions, rotation, and classification against ground-truth values.

Precise estimation of all these parameters is essential and this estimation often requires specialized strategies when using low-precision number representations for inference. Solving the challenges of converting a 3DOD network and deploying it on our inference chip using reduced-precision weights and activations enabled Recogni to fine-tune its approaches to various aspects of the conversion problem.

The results of this conversion appear in *Figure 2*. The top half of the figure shows a scene overlaid with 3D bounding boxes generated by the network running at 32-bit floating-point precision. The bottom half of the figure shows the 3D bounding boxes predicted by the Recogni 3DOD pipeline after optimization of the weights and activations using the three tips described in this article. The differences between the ground-truth and predicted bounding boxes aren't visible to the human eye, hence indicating a level of perfor-



3. Effective inference conversion combines batch norm folding (BNF), neural-network Cluster Compression (CC) techniques, and reduced-precision numeric representation including Recogni's custom logarithmic number system (LNS).

mance that's essential for AV applications.

The resulting model conversion strategies outlined in the following section can handle complex problems and are well-suited to many NN applications. The three tips discussed will help you strike a balance between compression rate and high accuracy.

Improve Neural-Network Efficiency via Model Conversion

Reducing the over-parameterization of the network by compressing its weights is a key pillar for inference deployment. For example, the weights of the 3DOD network presented earlier have been compressed from 240 MB to approximately 8.2 MB—a 29x compression ratio that makes it possible to store all of the weights on the same chip along with the computational machinery implementing the NN.

Most state-of-the-art NN architectures contain inefficiencies that can be optimized for inference deployment, including the elimination of over-parameterization, reducing the precision of overly precise computations, and removing unnecessary calculations. Recogni's chip is designed from the ground up to reduce the computational load of NN inference by exploiting all these inefficiencies.

Co-designing the hardware and the NN algorithm makes it possible to find the best tradeoff between accuracy, compute, memory, and power consumption by using batch norm folding, Cluster Compression techniques, and reduced-precision number representations. *Figure 3* illustrates the conversion process, and the following sections explain each conversion step in more detail.

Batch Norm Folding (BNF)

DNNs often contain operations such as batch normalization layers that aren't needed for inference tasks in deployed equipment. Implementing these batch normalization layers in hardware is expensive, especially when using floating-point precision, and it's not needed.

You can eliminate batch normalization layers during the conversion process by folding their scale and shifting parameters into the preceding layer's weights. This procedure is called batch norm folding (BNF) and has become a standard procedure when using deployed NNs for inference.

Cluster Compression (CC)

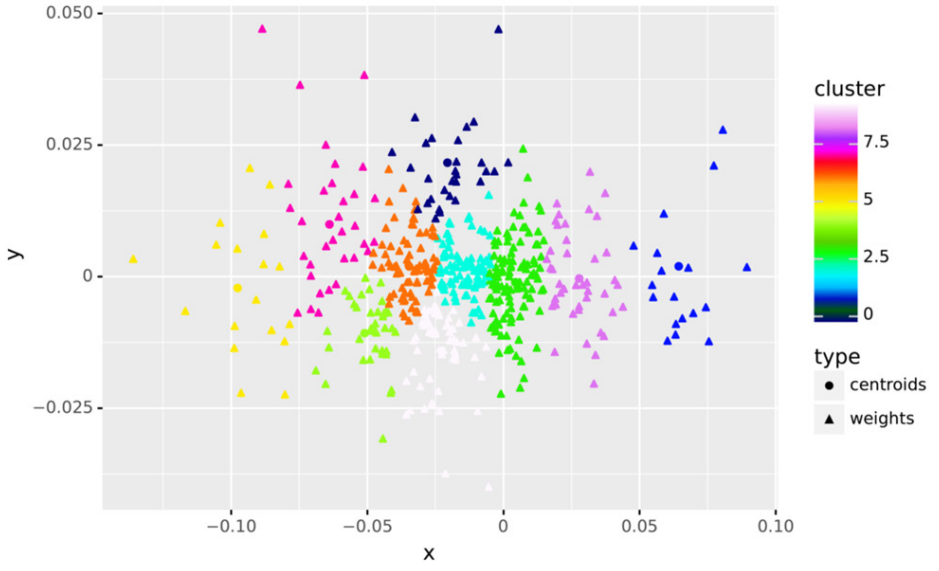
State-of-the-art DNN architectures require huge amounts of computing resources and memory for inferencing. Specialized NN accelerators used for inferencing must provide sufficient computing resources and feed them at high bandwidths to meet performance and latency requirements.

In many systems, it's memory—not computing capacity—that becomes the main bottleneck with respect to throughput and energy consumption. Specifically, accessing data in off-chip memory such as DRAM is extremely costly from both time and power-consumption perspectives. The closer that weights and activations can be brought to the computing resources, the better.

One effective way to reduce off-chip memory accesses is to compress weights so that they can all be stored on the accelerator chip itself. Many upcoming inferencing chips use compressed number formats such as FP8 (an 8-bit floating-point format) to reduce the amount of memory needed to store weights and activations. Reduced number formats are discussed in more depth in the next section of this article.

DNNs are known to be over-parameterized in general, so there's lots of opportunity to improve compression beyond shrinking the fundamental data type. For example, it's well known that the number of neural-network parameters can be significantly reduced by pruning channels or further compressing the weights with only a minor loss in model accuracy.

Recogni's perception accelerator technology reduces DNN memory requirements even further by using a proprietary compression scheme that stores compressed weights using a technique called Cluster Compression (CC).[1] Starting from a pre-trained model, CC extracts representative weight kernel centroids. Each centroid replaces the corresponding kernels of the same cluster *(Fig. 4)*, and then indexed representations are stored instead of saving whole kernels. Kernels in the same cluster share weights, which



4. Cluster Compression uses the k-means algorithm to create a compressed representation of each weight tensor. These weights can be stored in a lookup table, using just a pointer to index into the table.

provides significant additional compression.

After the initial clustering, the centroids can be further adjusted to improve accuracy by using standard back-propagation techniques. This additional layer of compression can make it possible to fit all required weights into on-chip memory, significantly reducing data movement and energy consumption.

Compressed Number System

This article has already discussed the use of compressed number systems such as FP8 to reduce memory storage requirements and cut power consumption. A recent whitepaper by Arm, Intel, and NVIDIA proposes a pair of standardized FP8 formats.[2] It shows that these reduced-precision formats can deliver comparable accuracy to 16-bit precision across a wide array of use cases, architectures, and networks while cutting storage requirements in half or into one quarter, compared to the 32-bit floating-point number system that's mostly used for training and evaluation of CNNs.

However, it's possible to take this idea even further to minimize the storage requirement of using a compressed number system. Most operations employed in modern DNNs, including convolutions, can be decomposed into mathematical primitives such as additions and multiplications.

Multiplications are expensive in terms of both hardware (die area on silicon) and power consumption, particularly for high-precision number formats such as 32-bit floating-point. The required chip area grows roughly quadratically with the bit size of the number format, so format compression is quite important when reducing the cost of inference.

Recogni's perception chip uses a logarithmic number system (LNS), which is an optimized FP8 variant long before the industry started picking it up as an improved evolutionary successor to the predominant INT8 number system.

The chip uses approximate math specifically tailored to NNs, which replaces costly multiplications in the linear domain with much simpler additions in the logarithmic domain. The same principle can be applied to convolutions. However, some calculations must still be done in the linear number domain rather than the logarithmic domain, so the Recogni perception chip has specialized hardware to convert between both domains.

Meeting Next-Gen Inferencing Demands for AVs

Reducing floating-point weights and activations to smaller number formats for inferencing applications is a commonly used method that allows NN developers to reduce storage and power-consumption requirements in inferencing applications.

However, it's possible to go much further with careful design of the inferencing hardware by designing the conversion to a compressed number format natively into silicon. This unlocks compute efficiencies of well over 15 TOPS/W in real applications, a common measurement unit to describe the efficiency of a chip design.

These steps for reducing computing, memory, and bandwidth requirements will be increasingly essential to meeting the demanding needs of next-generation inferencing applications for AVs.

References

1. S. Son, S. Nah, and K. M. Lee, "Clustering convolutional kernels to compress deep neural networks," Proceedings of the European conference on computer vision (ECCV), 2018, pp. 216-232.

2. Paulius Micikevicius, Dusan Stosic, Neil Burgess, et al, "FP8 Formats for Deep Learning," arXiv:2209.05433