Appendix: "Boosting the Throughput and Accelerator Utilization of Specialized CNN Inference Beyond Increasing Batch Size"

Jack Kosaian [1 *] **Amar Phanishayee** [2] **Matthai Philipose** [2] **Debadeepta Dey** [2] **K. V. Rashmi** [1]

A. Datasets

A.1. Datasets and models used in game-scraping application

This section provides details on the datasets and models used in the production video-game-scraping workload described in §2. The images in each dataset represent the style of text that will appear in a particular portion of a game screen, which will be used in a downstream event detection pipeline.

Dataset generation. The location and style of relevant text in a particular video game may differ from stream-to-stream. To avoid the need to manually label streams, the game-scraping application generates synthetic datasets for training, validation, and testing.

Specifically, the text that will appear in images for a particular dataset follows a predefined structure. For example, the text appearing in images of the V1 task is of the form "XY.Zk", where X, Y, and Z each represent a digit 0 through 9, and k is the string literal "k". From these specifications, examples that match certain classes of a particular dataset can be generated. For example, V1 classifies the Z digit in the specification above, and might generate "67.8k" and "04.8k" as instances of this specification for class "8".

Once an instance of a specification has been constructed, an image containing this text is generated. In order to train a model that is robust to perturbations in text location, text font, and background color/texture, the generation process selects fonts, locations, and backgrounds for the generated image at random from a set of prespecified options. Figures 1–6 below show the effects of this randomization.

We now provide details of each dataset used for this task in the paper. Example images chosen randomly from the validation sets of each dataset are displayed. We also describe the detailed architecture of the specialized CNNs employed for each dataset. For brevity, we use the following notation to describe CNNs: CX is a 3×3 2D convolution with X output channels and stride of 1, M is a 2D max pool with kernel size 3×3 and stride of 1, FX is a fully-connected layer with X output features. We use B as shorthand notation for C32 $\rightarrow$ C32 $\rightarrow$ C32. ReLUs follow each convolutional layer and all but the final fully-connected layer.

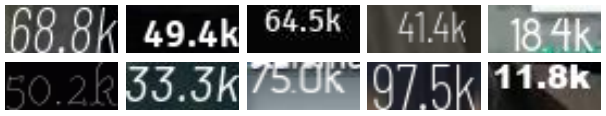


Figure 1. Example images in the lol-gold1 dataset.

V1: lol-gold1.

- **Game:** League of Legends
- **Number of classes:** 11
- **Image resolution:** (22, 52)
- **Example:** Figure 1
- **Model:** C32 $\rightarrow$ M $\rightarrow$ B $\rightarrow$ M $\rightarrow$ C8 $\rightarrow$ M $\rightarrow$ F11
- **Description:** Classifies the fractional value of a count of the amount of gold a player has accumulated (e.g., "7" in "14.7k"). Classes are digits 0 through 9 and "other" indicating that the section is blank.
- **Training images per class:** 10000
- **Validation images per class:** 100
- **Test images per class:** 100

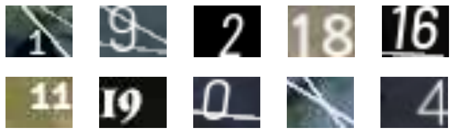


Figure 2. Example images in the apex-count dataset.

V2: apex-count.

- **Game:** Apex Legends
- **Number of classes:** 22
- **Image resolution:** (19, 25)
- **Example:** Figure 2

[*] Work done in part as an intern at Microsoft Research. [1] Carnegie Mellon University [2] Microsoft Research. Correspondence to: Jack Kosaian <jkosaian@cs.cmu.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

- **Model:** C32 → M → B → M → C8 → M → F22
- **Description:** Classifies the number of members of a squad remaining. Classes are integers 0 through 20 and "other" indicating that the section is blank.
- **Training images per class:** 1000
- **Validation images per class:** 100
- **Test images per class:** 100

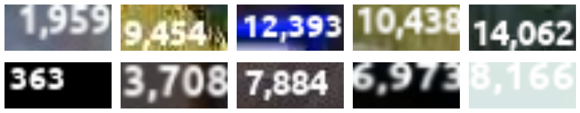


Figure 3. Example images in the sot-coin dataset.

V3: sot-coin.

- **Game:** Sea of Thieves
- **Number of classes:** 15
- **Image resolution:** (17, 40)
- **Example:** Figure 3
- **Model:** C32 → M → B → M → C8 → M → F15
- **Description:** Classifies the thousands-place of a count on the number of coins a player has (e.g., "10" for "10,438"). Classes are integers 0 through 14 and "other" indicating that the section is blank.
- **Training images per class:** 800
- **Validation images per class:** 200
- **Test images per class:** 200

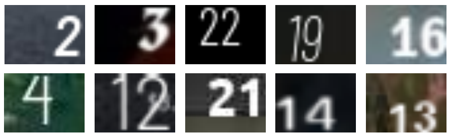


Figure 4. Example images in the sot-time dataset.

V4: sot-time.

- **Game:** Sea of Thieves
- **Number of classes:** 27
- **Image resolution:** (22, 30)
- **Example:** Figure 4
- **Model:** C32 → M → B → M → B → M → C8 → M → F27
- **Description:** Classifies the time remaining. Classes are integers 0 through 25 and "other" indicating that the section is blank.
- **Training images per class:** 2000
- **Validation images per class:** 100
- **Test images per class:** 100

V5: lol-gold2.

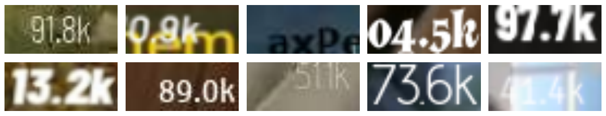


Figure 5. Example images in the lol-gold2 dataset.

- **Game:** League of Legends
- **Number of classes:** 111
- **Image resolution:** (22, 52)
- **Example:** Figure 5
- **Model:** C32 → M → B → M → C8 → M → F111
- **Description:** Classifies the integer value of a count of the amount of gold a player has accumulated (e.g., "14" in "14.7k"). Classes are digits 0 through 9, 00 through 99, and "other" indicating that the section is blank.
- **Training images per class:** 1000
- **Validation images per class:** 100
- **Test images per class:** 100
- **Note:** This CNN is used only in §I

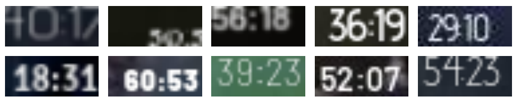


Figure 6. Example images in the lol-time dataset.

V6: lol-time.

- **Game:** League of Legends
- **Number of classes:** 62
- **Image resolution:** (15, 35)
- **Example:** Figure 6
- **Model:** C32 → M → B → M → C8 → M → F62
- **Description:** Classifies the minutes place of a timer (e.g., "30" in "30:54"). Classes are digits 00 through 60 and "other" indicating that the section is blank.
- **Training images per class:** 1000
- **Validation images per class:** 100
- **Test images per class:** 100
- **Note:** This CNN is used only in §I

A.2. Datasets and models used in NoScope

We evaluate folding using four of the datasets from NoScope (Kang et al., 2017). Each task involves binary classification of whether an object of interest is present in a frame. As the overall videos provided contain millions of frames, we sample contiguous subsets of frames in the video to form training, validation, and testing sets.

The CNNs used in evaluation follow those described in the NoScope paper and source code. We next detail these architectures, as well as the splits of the dataset used in evaluation.

N1: coral.

- **Object of interest:** person
- **Model:** C16 $\rightarrow$ C16 $\rightarrow$ M $\rightarrow$ F128 $\rightarrow$ F2
- **Total video duration:** 11 hrs.
- **Dataset split:** Split the video into eight contiguous chunks. Use chunk 6 as a training dataset, chunk 7 as a validation dataset, and chunk 8 as a testing dataset.

N2: night.

- **Object of interest:** car
- **Model:** C16 $\rightarrow$ C16 $\rightarrow$ M $\rightarrow$ F128 $\rightarrow$ F2
- **Total video duration:** 8.5 hrs.
- **Dataset split:** Split the video into eight contiguous chunks. Use chunk 2 as a training dataset, chunk 3 as a validation dataset, and chunk 4 as a testing dataset.

N3: roundabout.

- **Object of interest:** car
- **Model:** C32 $\rightarrow$ C32 $\rightarrow$ M $\rightarrow$ C64 $\rightarrow$ C64 $\rightarrow$ M $\rightarrow$ F32 $\rightarrow$ F2
- **Total video duration:** 8.1 hrs.
- **Dataset split:** Split the video into eight contiguous chunks. Use chunk 2 as a training dataset, chunk 3 as a validation dataset, and chunk 4 as a testing dataset.

N4: taipei.

- **Object of interest:** bus
- **Model:** C64 $\rightarrow$ C64 $\rightarrow$ M $\rightarrow$ F32 $\rightarrow$ F2
- **Total video duration:** 12 hrs.
- **Dataset split:** Split the video into sixteen contiguous chunks. Use the chunk 1 as a training dataset, chunk 2 as a validation dataset, and chunk 3 as a testing dataset.

B. Example of Folding a Single Layer

Table 1 shows an example of folding a convolutional layer from a specialized CNN used in the game-scraping workload. The memory traffic of the original layer is dominated by the input and output activations of the layer. Folding with $f = 4$ reduces memory traffic by nearly 2× while maintaining the same number of operations, enabling a 2× increase in arithmetic intensity.

C. Folding for Group Convolutions

In this section, we describe how folding is applied to group convolutions.

Background on group convolutions. In a group convolution, the input and output channels of the convolution are split into G groups. Each output channel in a particular group is computed via convolution over only those input channels in the corresponding group. This results in a G-fold decrease in operations and a G-fold decrease in the number of parameters in the convolutional layer. The resultant arithmetic intensity for a group convolution is thus:

$$\frac{2NHWC_oC_iK_HK_W/G}{B(NHWC_i + \frac{C_iK_HK_WC_o}{G} + NHWC_o)}$$

The arithmetic intensity of a group convolution in the batch-limited regime (defined in §2.3) is determined as follows (recalling from §2.3 that calculating arithmetic intensity in the batch-limited regime involves removing the variable B):

$$A = \frac{2NHWC_oC_iK_HK_W/G}{NHWC_i + \frac{C_iK_HK_WC_o}{G} + NHWC_o}$$

$$\lim_{N\to\infty} A = \frac{2C_oC_iK_HK_W}{C_i + C_o} * \frac{1}{G}$$

Comparing this arithmetic intensity to that in Eqn. 4 of the main paper, the arithmetic intensity of a group convolution with G groups in the batch-limited regime is $G\times$ lower than a corresponding "vanilla" convolution. This makes group convolutions a promising target for increasing arithmetic intensity via folding.

Applying folding to group convolutions. Folding group convolutions is straightforward. Similar to folding "vanilla" convolutions, a FoldedCNN for a group convolution with C_i input channels and C_o output channels reduces batch size by a factor of f and increases C_i and C_o each by a factor of $\sqrt{f}\times$. This results in increasing the number of channels per group in the group convolution by a factor of $\sqrt{f}$, and thus also increases arithmetic intensity in the batch-limited regime by a factor of $\sqrt{f}$.

Inference performance of folded group convolutions. We evaluate the throughput and utilization of folding on two group convolutions shown in Table 2. The two group convolutions are identical other than the number of total input and output channels, with G32 having 32 and G64 having 64. Each setting uses 4 groups, leading to 8 and 16 channels per group for G32 and G64, respectively. We compare the throughput and utilization of these convolutions to the corresponding folded version with $f = 4$. Folding results in 64 input and output channels with 16 channels per group for G32, and 128 input and output channels with 32 channels per group for G64. We use the same experimental setup described in §5 of the main paper for evaluating inference performance.

With batch size of 1024, folding with $f = 4$ increase throughput and utilization of these grouped convolutions

Table 1. Example of increasing arithmetic intensity by folding a convolutional layer with $f = 4$. The layer has $K_H = K_W = 3$, $H = 11$, $W = 26$, and uses half precision (i.e., $B = 2$).

	Original		**Folded (f = 4)**	
	Equation	**Value**	**Equation**	**Value**
Batch size	N	1024	N/f	256
Input, output channels	C_i, C_o	32, 32	$C_i\sqrt{f}, C_o\sqrt{f}$	64, 64
Input and output elements (E_{io})	$NHWC_i + NHWC_o$	18.74M	$\frac{\sqrt{f}}{f}NHWC_i + \frac{\sqrt{f}}{f}NHWC_o$	9.37M
Layer elements (E_l)	$C_iK_HK_WC_o$	0.01M	$fC_iK_HK_WC_o$	0.04M
Memory traffic in bytes (M)	$B(E_{io} + E_l)$	37.51M	$B(E_{io} + E_l)$	18.82M
Operations (O)	$2NHWC_oC_iK_HK_W$	5398.07M	$2NHWC_oC_iK_HK_W$	5398.07M
Arithmetic intensity	O/M	**143.93**	O/M	**286.87**

Table 2. Group convolutions evaluated

Name	C_i	C_o	G	K_H	K_W	H	W
G32	32	32	4	3	3	50	50
G64	64	64					

by 1.74× for G32 and by 1.59× for G64 on a V100 GPU. Folding increases arithmetic intensity by nearly a factor of two for each convolution. The larger improvement for G32 compared to G64 comes from the lower arithmetic intensity of G32; due to having half the number of input and output channels of G64, G32 has half the arithmetic intensity. Thus, there is more room for improving the utilization of G32 by increasing arithmetic intensity alone via folding. These results show the effectiveness of folding on group convolutions.

D. Folding for Winograd Convolutions

FoldedCNNs can benefit a wide variety of convolutional implementations, such as direct convolutions, matrix-multiplication-based convolutions, and Winograd convolutions. In fact, our evaluation in §5 runs atop TensorRT, which selects among convolutional implementations, including Winograd. To more clearly illustrate the performance of FoldedCNNs on Winograd convolutions, we also directly run FoldedCNNs using Winograd convolutions in cuDNN. Here, on the video scraping CNNs using the same experimental setup described in §5, FoldedCNNs with $f = 4$ provided a median speedup of 1.66× over the original CNN, matching the speedups in §5.3.

E. Inference Performance on T4 GPU

Figure 7 shows the throughput, FLOPs/sec, and arithmetic intensity achieved by FoldedCNNs and the original CNN on a T4 GPU (AWS g4dn.xlarge instance) in half-precision. The general trends are similar to those described in §5 of the main paper for the V100 GPU.

F. Speedup with Varying Batch Size

Figure 8 shows the throughput improvement when using FoldedCNNs with various values of f relative to the original CNN at varying batch sizes. As shown in the figure, the throughput improvement resulting from folding is largest at a batch size of 2048, and decreases with decreasing batch size. This behavior is expected, as decreasing batch size N decreases the likelihood that the inequality proved in §J will hold, and thus that folding will benefit. Folding is designed for improving high-throughput specialized CNN inference, in which large batch sizes are used.

G. Accuracy-Throughput Tradeoff

When reasoning about the potential tradeoff between accuracy and throughput/utilization present with FoldedCNNs, it is important to consider the usecases of specialized CNNs. As described in §2.1, it is common to use specialized CNNs as a lightweight filter in front of a large, general-purpose CNN. In such systems, most inputs are processed only by the specialized CNN, rather than by both the specialized CNN and the general-purpose CNN. Thus, the throughput of the specialized CNN typically dominates the total throughput of the system.

Given the heavy use of the specialized CNN in this setup, improving the throughput of the specialized CNN at the expense passing more inputs to the general-purpose CNN may increase system throughput. For example, a FoldedCNN with $f = 4$ speeds up the N2 CNN by 2.50× with a 0.21% drop in accuracy. We show below that this FoldedCNN increases system throughput unless the general-purpose CNN is over 285× slower than the N2 CNN. Thus, the improved utilization and throughput of specialized CNNs made possible by FoldedCNNs can compensate for reduced their accuracy to improve total system throughput.

We now walk through this accuracy-throughput tradeoff via an abstract example. Figure 9 shows an abstract example of using a specialized CNN (e.g., those from NoScope) as a lightweight filter in front of a large, general-purpose CNN (e.g., ResNet-50). As depicted in the figure, all inputs pass

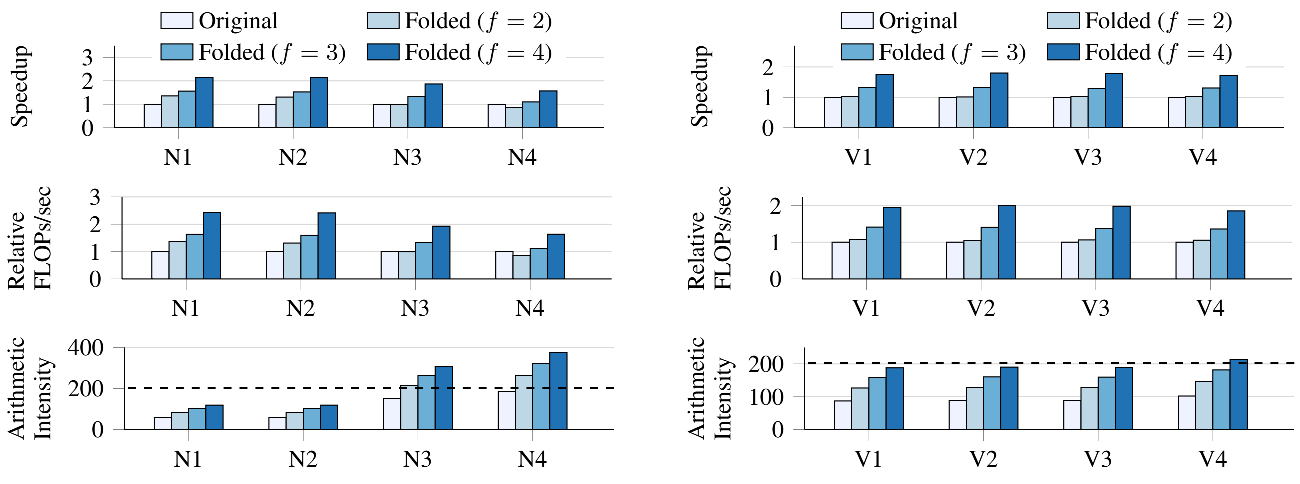


Figure 7. Inference performance of FoldedCNNs relative to the original CNN. Arithmetic intensity is plotted in absolute numbers, and the dashed line shows the minimum arithmetic intensity required to reach peak FLOPs/sec on a T4 GPU.

through the specialized CNN, which has a latency of T_s. The specialized CNN is unsure about u fraction of those inputs, and thus forwards the inputs to the general-purpose CNN, which has a latency of T_g. For the remaining $(1-u)$ fraction of inputs, the specialized CNN is sure of its answer, and returns the prediction directly.

The expected latency for a given input to this system is thus:

$$E[T] = T_s + uT_g$$

Suppose that one replaced the specialized CNN used in such an application with a FoldedCNN that increases throughput by a factor of x, but decreases accuracy by a. Under the reasonable assumption that an increase in throughput leads to a corresponding decrease in latency, the latency of the FoldedCNN can be given as $\frac{T_s}{x}$. Furthermore, under the assumption that all incorrectly classified inputs from the specialized CNN are forwarded to the general-purpose CNN (i.e., u is equivalent to the error of the specialized CNN), then the FoldedCNN lets $u + a$ fraction of frames through to the general-purpose CNN. Thus, the expected latency for a given input to the system with a FoldedCNN is:

$$E[T] = \frac{T_s}{x} + (u+a)T_g$$

Clearly, for high values of x and small values a, the FoldedCNN can result in improved total system throughput (reciprocal of latency). A secondary question of interest is: given specific values of x and a, for what values of T_s and T_g does the FoldedCNN increase overall system throughput?

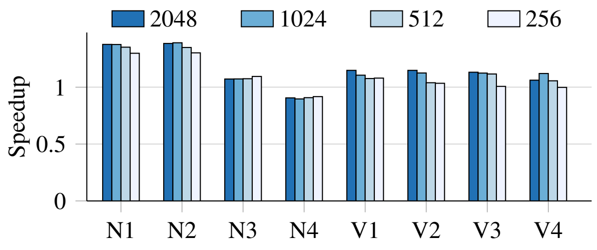
To answer this question, we focus on the ratio $\frac{T_g}{T_s}$. Intuitively, the higher this ratio, the larger the effect of inaccuracy of the FoldedCNN on overall system throughput. We next calculate the maximum value this ratio can be for a FoldedCNN to improve overall system throughput:

$$T_s + uT_g > \frac{T_s}{x} + (u+a)T_g$$
$$T_s - \frac{T_s}{x} > (u+a)T_g - uT_g$$
$$T_s(1 - \frac{1}{x}) > aT_g$$
$$\frac{1}{a}(1 - \frac{1}{x}) > \frac{T_g}{T_s}$$

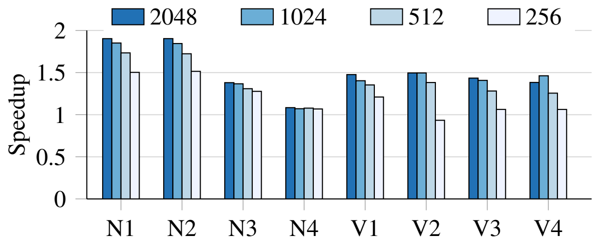
Consider the FoldedCNN with $f = 4$ for the N2 dataset described in §5.2 of the main paper. This FoldedCNN results in an increase in throughput of $x = 2.5\times$ and a decrease in accuracy of $a = 0.0021$. Plugging these values into the inequality above shows that this FoldedCNN will result in an overall improvement in system throughput so long as the general-purpose CNN is less than 285× slower than the original specialized CNN. If we consider ResNet-50 as an example of a general-purpose CNN, this is easily satisfied for the N2 CNN: ResNet-50 is 83× slower than the original specialized CNN.

H. Effect of Tile Quantization on the Performance of FoldedCNNs

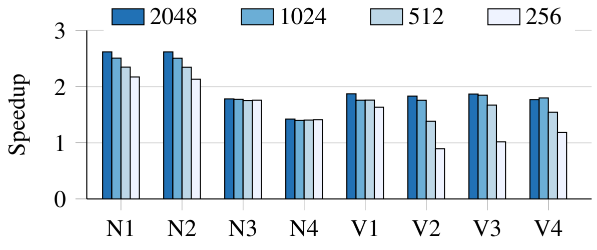
In §5.2, we observed one case in which a FoldedCNN resulted in a decrease in throughput and utilization compared to the original CNN: the N4 CNN using $f = 4$. After investigating the CNN, we found the cause to be due to *GPU tile quantization*: when the problem size does not divide evenly into a chosen tile size (i.e., the size of partitions of the overall kernel) (NVIDIA). NVIDIA's deep learning libraries are best optimized for cases in which certain parameters of a convolutional layer, such as input and output channels, are



(a) $f = 2$



(b) $f = 3$



(c) $f = 4$

Figure 8. Speedup of FoldedCNNs with varying f at various batch sizes relative to the throughput of the original CNN at corresponding batch sizes.

divisible by large powers of two (e.g., divisible by 64 or 128) (NVIDIA). Parameters that do not meet this requirement typically use kernels optimized for the next highest number divisible by a large power of two, resulting in a significant amount of wasted work. For more details on the inefficiency resulting from tile quantization, please see NVIDIA's deep learning performance guide (NVIDIA).

Many CNNs are already designed to have a number of input and output channels that are a power of two (e.g., many of the specialized CNNs have convolutional layers with 32 input and output channels). However, FoldedCNN's increase the number of input and output channels by a factor of $\sqrt{f}$. For non-square values of f, such as 2 and 3, applying folding to such a layer may result in a number of input or output channels that is no longer a power of two or is no longer divisible by a power of two. For example, applying folding with $f = 2$ to a convolutional layer with 64 input and output channels will result in a convolutional layer with $\lfloor 64\sqrt{2} \rfloor = 90$ channels.

For the values of f considered in this work, we find that tile

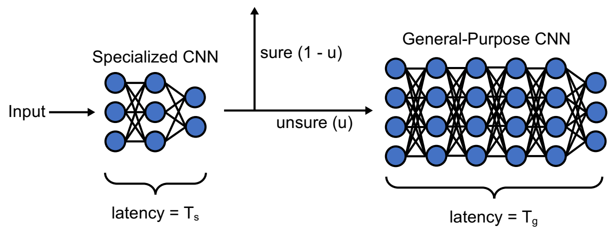


Figure 9. Abstract example of the use of a specialized CNN as a lightweight filter in front of a larger, general-purpose CNN.

quantization primarily affects convolutions with a number of input and output channels greater than or equal to 64; we do not observe the negative effects often associated with tile quantization for convolutions with fewer channels, such as 32 or 16.

As shown in §A, the N4 CNN contains two convolutions with 64 intermediate channels, followed by a fully-connected layer with 32 output neurons. The FoldedCNN with f of 2 and 3 will thus lead to the negative effects of tile quantization for the convolutions in this CNN, but not for the fully-connected layer, which will receive the full benefits of folding. With $f = 2$, the benefit from folding does not outweigh the inefficiency due to tile quantization, resulting in a net decrease in utilization and throughput. In contrast, with $f = 3$, the benefits of folding outweigh the cost of tile quantization, resulting in an increase in utilization and throughput, albeit less pronounced than expected for $f = 3$.

It is important to note that this case with decreased utilization and throughput is *not due to incorrectness of the transformations performed by FoldedCNNs.* As shown in §5.2 of the main paper, FoldedCNNs with f of 2 and 3 for the N4 CNN result in the expected $\sqrt{f}\times$ improvements in arithmetic intensity. Decreased inference performance in this case is due to lower levels of the system software (e.g., TensorRT, cuDNN), rather than the design of FoldedCNNs themselves. Other accelerators may not face the same issue.

I. Evaluation on Small CNNs with many Classes

In this section, we consider CNNs that have the same size as specialized CNNs, but which operate over many classes. We consider two new game-scraping tasks: a task with 111 classes (V5), and one with 62 classes (V6[1]). We use the same CNN as that used for V1. Table 3 shows that FoldedCNNs exhibit larger drops in accuracy on these tasks due

[1]For this CNN, we find that the small input resolution and large number of classes requires using more specially-tuned curriculum learning parameters. Specifically, when training a FoldedCNN with $f = 4$ on this dataset, we use $I = 4$, $\Delta = 3$, and $E = 120$, and train the CNN for 3000 epochs.

Table 3. Performance of FoldedCNNs for CNNs with many classes. Differences in accuracy are listed in parentheses.

			Original	**Folded (f = 2)**		**Folded (f = 3)**		**Folded (f = 4)**	
Model	**Resolution**	**Classes**	**Accuracy**	**Accuracy**	**Speedup**	**Accuracy**	**Speedup**	**Accuracy**	**Speedup**
V5	(22, 52)	111	93.95	92.50 (-1.45)	1.12	90.78 (-3.17)	1.42	87.51 (-6.44)	1.75
V6	(15, 35)	62	89.71	88.03 (-1.68)	1.08	85.48 (-4.23)	1.42	84.92 (-4.79)	1.71

to the larger number of classes, but still increase utilization/throughput by up to 1.75×.

J. Proof of Reduction in Memory Traffic

We will prove that the transformation described in §3 reduces total memory traffic if:

$$NHW > \frac{(f-1)C_iK_HK_WC_o}{(1-\frac{1}{\sqrt{f}})(C_i+C_o)} \quad (1)$$

Recall that the arithmetic intensity of a convolutional layer as given by Eqn. 3 is:

$$\frac{\text{FLOPs}}{\text{Bytes}} = \frac{2NHWC_oC_iK_HK_W}{B(NHWC_i + C_iK_HK_WC_o + NHWC_o)}$$

In §3, we create a new version of the layer in which we decrease NHW by a factor of f and increase both C_i and C_o by a factor of $\sqrt{f}$. We wish to show that this new layer has a higher arithmetic intensity than the original layer, provided that:

$$NHW > \frac{(f-1)C_iK_HK_WC_o}{(1-\frac{1}{\sqrt{f}})(C_i+C_o)}$$

We will first show that the new layer performs an equal number of operations as the original layer (i.e., the numerator in Eqn. 3 stays the same) and then show that the new layer layer has reduced memory traffic compared to the original layer (i.e., the denominator in Eqn. 3 decreases), provided that the inequality holds. These two changes will result in the new layer having an increased arithmetic intensity.

Equal number of operations. The initial convolutional layer performs $2NHWC_oC_iK_HK_W$ operations. The transformed convolutional layer performs $\frac{2NHW}{f}(\sqrt{f}C_o)(\sqrt{f}C_i)(K_HK_W) = 2NHWC_oC_iK_HK_W$ operations, which is equal to that of the original model.

Reduced memory traffic. We wish to show that the inequality is equivalent to the memory traffic of the transformed layer being lower than that of the original layer.

We first note that, ignoring the bytes per element B, the memory traffic of the original convolutional layer is $NHWC_i + C_iK_HK_WC_o + NHWC_o$, while that of the transformed layer is $\frac{\sqrt{f}}{f}NHWC_i + fC_iK_HK_WC_o + \frac{\sqrt{f}}{f}NHWC_o$.

We wish to show that:

$$NHWC_i + C_iK_HK_WC_o + NHWC_o > \frac{\sqrt{f}}{f}NHWC_i + fC_iK_HK_WC_o + \frac{\sqrt{f}}{f}NHWC_o$$

We first rearrange the righthand side of the inequality as:

$$\frac{\sqrt{f}}{f}NHWC_i + fC_iK_HK_WC_o + \frac{\sqrt{f}}{f}NHWC_o = \frac{NHW}{\sqrt{f}}C_i + fC_iK_HK_WC_o + \frac{NHW}{\sqrt{f}}C_o$$

Grouping by similar terms gives:

$$NHWC_i - \frac{NHW}{\sqrt{f}}C_i + NHWC_o - \frac{NHW}{\sqrt{f}}C_o > fC_iK_HK_WC_o - C_iK_HK_WC_o$$

Which implies:

$$(1-\frac{1}{\sqrt{f}})NHWC_i + (1-\frac{1}{\sqrt{f}})NHWC_o > (f-1)C_iK_HK_WC_o$$

Which implies:

$$NHW((1-\frac{1}{\sqrt{f}})(C_i+C_o)) > (f-1)C_iK_HK_WC_o$$

Which ultimately leads to our desired inequality:

$$NHW > \frac{(f-1)C_iK_HK_WC_o}{(1-\frac{1}{\sqrt{f}})(C_i+C_o)}$$

References

Kang, D., Emmons, J., Abuzaid, F., Bailis, P., and Zaharia, M. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proceedings of the VLDB Endowment*, 10 (11):1586–1597, 2017.

NVIDIA. NVIDIA Deep Learning Performance Guide. https://docs.nvidia.com/deeplearning/sdk/dl-performance-guide/index.html. Last accessed 08 June 2021.