Rapid Exact Signal Scanning with Deep Convolutional Neural Networks

Markus Thom and Franz Gritschneder

***Abstract*—A rigorous formulation of the dynamics of a signal processing scheme aimed at dense signal scanning without any loss in accuracy is introduced and analyzed. Related methods proposed in the recent past lack a satisfactory analysis of whether they actually fulfill any exactness constraints. This is improved through an exact characterization of the requirements for a sound sliding window approach. The tools developed in this paper are especially beneficial if Convolutional Neural Networks are employed, but can also be used as a more general framework to validate related approaches to signal scanning. The proposed theory helps to eliminate redundant computations and renders special case treatment unnecessary, resulting in a dramatic boost in efficiency particularly on massively parallel processors. This is demonstrated both theoretically in a computational complexity analysis and empirically on modern parallel processors.**

***Index Terms*—Deep learning techniques, dense signal scanning, sliding window approach, convolutional neural networks.**

I. Introduction

EVEN though today's signal processing systems have achieved an unprecedented complexity, a multitude of them have a very basic commonality: The application of a translation-invariant function to a large signal in a sliding fashion facilitates the dense computation of interesting output values for each possible spatial location. Consider filter-based signal denoising as an example: Here, each entry of the denoised output signal always depends on a fixed computation rule applied to only a limited number of samples within the input signal, or in other words, on a *subsignal* of the input signal. The computation rule is completely agnostic with regard to the actual position, it is merely important that the input samples are drawn accordingly from the input signal.

Of course, modern systems apply more sophisticated techniques than mere filtering. However, recently an architecture essentially made up of simple filtering building blocks has displayed advantages over any other approach in a wide variety of practical applications. Due to significant advances in the design of massively parallel processors and the availability of huge annotated data sets, deep artificial neural networks which learn desired behavior by adapting their degrees of freedom to concrete sample data rather than being programmed explicitly have become the de facto state-of-the-art in the domains of signal restoration and signal classification.

The most important architecture for analyzing signals that possess a spatial structure, such as images where pixels are arranged on a two-dimensional grid, was inspired by findings on the dynamics of mammalian visual cortex [1]: *Convolutional Neural Networks (CNNs)* [2], [3], [4] respect the weight-sharing principle, hence convolution with trainable filters becomes the actual workhorse for data processing. This principle greatly reduces the network's degrees of freedom, making it less susceptible to overfitting, and it incorporates a strong prior with respect to the spatial layout of the input data. In fact, this particular architecture has proven highly successful both for image restoration tasks [5], [6], [7] and pattern recognition problems [8], [9], [10].

If a CNN trained for object categorization is evaluated at each feasible image position, it is possible to assign class membership estimations to all the pixels in an image, yielding a semantic segmentation of a scene [11], [12], [13]. This representation is much more powerful than what can be gained from a strict conventional object detection approach which solely outputs bounding boxes of found object instances. Instead, it facilitates applications such as automated biological or medical image analysis [11], [14], [15] and dense vehicle environment perception [16]. While the computational complexity of a sophisticated classification system used in conjunction with a sliding window approach may seem excessive at first glance, the weight-sharing principle of a CNN can be exploited so that intermediate computation results can be shared among adjacent image patches, resulting in a speedup of several orders of magnitude. Although this was already realized for CNNs without pooling layers more than two decades ago [17], approaches that also account for pooling layers emerged only recently [14], [18], [19].

The approach of Giusti *et al.* [14] achieves fast scanning of entire images through the introduction of a fragmentation data structure. Here, the internal representations of a CNN are decomposed using a spatial reordering operation after each pooling layer, allowing the evaluation of convolutions on contiguous signals at all times. The intermediate signals are however inhomogeneous with respect to their dimensionality, leaving the possibility for the use of efficient tensor convolution routines unclear. Li *et al.* [18], on the other hand, propose enlarging the filter banks of convolutional layers by inserting

Manuscript received October 5, 2015; revised February 27, 2016 and July 9, 2016; accepted October 9, 2016. Date of publication XXXXXXXX XX, XXXX; date of current version XXXXXXXX XX, XXXX. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Gustau Camps-Valls. This work was supported by Daimler AG, Germany.

M. Thom was with driveU / Institute of Measurement, Control and Microtechnology, Ulm University, 89081 Ulm, Germany. He is now with Daimler AG, 89081 Ulm, Germany (e-mail: markus.thom@daimler.com).

F. Gritschneder is with driveU / Institute of Measurement, Control and Microtechnology, Ulm University, 89081 Ulm, Germany (e-mail: franz.gritschneder@uni-ulm.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier XX.XXXX/TSP.XXXX.XXXXXXX

vanishing entries at regular locations. These sparse filter banks require a cumbersome re-engineering of efficient convolution implementations, which may not be able to achieve maximum throughput on modern massively parallel processors. Sermanet *et al.* [19] use the same processing pipeline for patches and entire images, which incurs relaxations with accuracy loss effects where the actual impact is hard to predict.

All these approaches have in common that it is not inherently clear what they actually compute or if the result is even the desired one. Instead of a rigorous mathematical proof of correctness, only toy examples are available, illustrating the implementation of these approaches. This situation is especially unsatisfactory if, instead of pure convenience functions, systems subject to safety considerations should be realized where precise statements rather than only an empirical evaluation are required.

The key contributions of this paper are (i) the development of an original theory on *subsignal compatible transformations* as exact characterization of functions that fulfill the invariants required for a sound sliding window approach, (ii) the proposition of a method for dense signal scanning *provably without any accuracy loss* that yields significant speedups due to homogeneous data structures and elimination of redundant computations and special case treatment, and (iii) the demonstration how CNNs interconnect with the theory and how they can be *exactly* transformed from subsignal-based application to signal-based application *without* any necessary adjustments to the computationally most demanding tensor convolution. To the authors' best knowledge, they are the first to actually have mathematically rigorous statements to support their claims on the correctness of dense signal scanning with CNNs. Due to the generality of the results, the herein developed theoretical framework can also serve as a basis for analyzing related and emerging methods for signal processing based on translation-invariant functions applied in a sliding fashion.

The remainder of this paper is structured as follows. Section II presents an introduction to the CNN structure, fixes the notation and introduces what is meant by subsignals. Section III establishes the basics of the theory on subsignal compatible transformations and shows how the building blocks of CNNs fit into the theory. In the following Sect. IV, the theory is extended to functions applied in a strided fashion, which is particularly important for pooling operators evaluated on non-overlapping blocks. Section V provides a theoretical computational complexity analysis. Practical considerations for image processing and the results of experiments on real parallel processors are discussed in Sect. VI. The paper is concluded with a discussion of the results in Sect. VII.

II. Prerequisites

This section begins with an introduction to the building blocks of a CNN. Then the notation used throughout the paper is established. The section concludes with the definition of the subsignal extraction operator and statements on its properties.

A. Convolutional Neural Networks

CNNs are organized in a number of specialized layers [4]. Each layer receives input data from its predecessor, processes it, and sends the result to the next layer. The network's output is then the output of the final layer. The training process consists of tuning the network's degrees of freedom until the network produces the desired output given concrete input sample data [20]. After a network has been trained, it can be used as a predictor on previously unseen data in regression or classification tasks.

The different specialized layer types are given as follows. *Convolutional layers* respect the weight-sharing principle: They convolve their input with a trainable filter bank and add a trainable scalar bias to form the layer output. These layers fall into the class of subsignal compatible transformations detailed in Sect. III, a mathematical analysis of the involved computations is given in Sect. III-C.

Fully-connected layers are a special case of convolutional layers in that they carry out a convolution with unit spatial filter size. Mathematical treatment of these layers is hence superseded by the analysis of convolutional layers.

Non-linearity layers independently pass each sample of a signal through a scalar transfer function. This prevents the entire network from forming a purely linear system and hence enhances the network's representational capacity. Since these operations are agnostic with respect to any spatial structure, an analysis is straightforward and handled in Sect. III-C.

Eventually, *pooling layers* strengthen a network's invariance to small translations of the input data by evaluation of a fixed pooling kernel followed by a downsampling operation. For brevity of the presentation, only functions applied to non-overlapping blocks are considered here. Pooling requires an extension of the plain theory of subsignal compatible transformations, provided in Sect. IV.

This paper proves that CNNs can be transformed from subsignal-based application to signal-based application by transforming strided function evaluation into sliding function evaluation and inserting special helper layers, namely fragmentation, defragmentation, stuffing and trimming. This transformation is completely lossless, both subsignal-based application and signal-based application lead to the same results. Even after the transformation, CNNs can be further fine-tuned with standard optimization methods. An example for this process is given in Sect. VI.

B. Notation

For the sake of simplicity, the mathematical analysis is restricted to vector-shaped signals. The generalization to more complex signals such as images is straightforward through application of the theory to the two independent spatial dimensions of images. This is briefly discussed in Sect. VI.

$\mathbb{N}_1 := \mathbb{N} \setminus \{ 0 \}$ represents the positive natural numbers. If M is a set and $q \in \mathbb{N}_1$, then M^q denotes the set of all q-tuples with entries from M. The elements of M^q are called *signals*, their q entries are called *samples*. If $\xi = (\xi_1, \dots, \xi_q) \in M^q$ is a signal and $I \in \{1, \dots, q\}^r$ is an index list with r entries, the *formal sum* $\omega := \sum_{\nu=1}^{r} \xi_{I_\nu} \cdot e_\nu^r$ is used for the element $\omega \in M^r$ with $\omega_\nu = \xi_{I_\nu}$ for all $\nu \in \{1, \dots, r\}$. For example, when M equals the set of real numbers $\mathbb{R}$ and hence M^r is the r-dimensional Euclidean space, then the formal sum

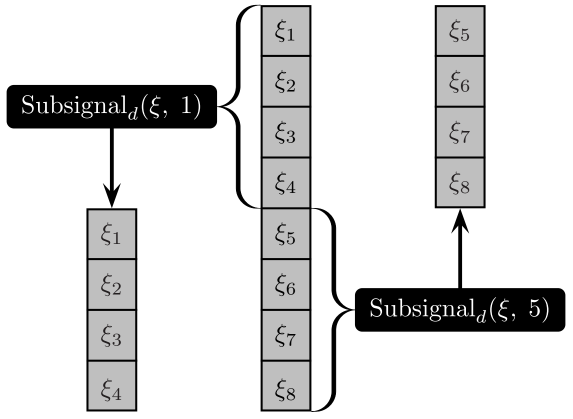


Fig. 1. Illustration of the subsignal extraction operator applied to a signal ξ with $D = 8$ samples for extraction of subsignals with $d = 4$ samples. The left-hand side shows the first subsignal and the right-hand side shows the final subsignal with the maximum subsignal index of $D - d + 1 = 5$.

ω corresponds to the linear combination of canonical basis vectors e_ν^r weighted with selected coordinates of the signal ξ.

For $\xi \in M^q$, $\dim_M(\xi) = q$ represents the *dimensionality* of ξ. This does not need to correspond exactly with the concept of dimensionality in the sense of linear algebra. If for example $M = \mathbb{N}^c$ for categorical data with $c \in \mathbb{N}_1$ features, then M^q is not a vector space over M. The theory presented in this paper requires algebraic structures such as vector spaces or analytic structures such as the real numbers only for certain examples. The bulk of the results hold for signals with samples from arbitrary sets.

If M is a set and $c \in \mathbb{N}_1$ is a positive natural number, then $\cup_c(M) := \cup_{q=c}^{\infty} M^q$ is written for the set that contains all the signals of dimensionality greater than or equal to c with samples from M. For example, if $\xi \in \cup_c(M)$ then there is a natural number $q \geq c$ so that $\xi = (\xi_1, \ldots, \xi_q)$ with $\xi_\nu \in M$ for all $\nu \in \{1, \ldots, q\}$. Note that $\cup_1(M)$ contains all non-empty signals with samples from M.

C. Division of a Signal into Subsignals

A *subsignal* is a contiguous list of samples contained in a larger signal. First, the concept of extracting subsignals with a fixed number of samples from a given signal is formalized:

Definition 1. Let M be an arbitrary set and let $d \in \mathbb{N}_1$ denote a fixed subsignal dimensionality. Then the function $\text{Subsignal}_d \colon \bigcup_{D=d}^{\infty} \left(M^D \times \{1, \ldots, D - d + 1\}\right) \to M^d$,

$$(\xi,\ i) \mapsto \sum\nolimits_{\nu=1}^{d} \xi_{i+\nu-1} \cdot e_\nu^d,$$

is called the *subsignal extraction operator*. Here, ξ is the input signal and i denotes the *subsignal index*.

It is straightforward to verify that Subsignal_d is well-defined and actually returns all possible $D - d + 1$ contiguous subsignals of length d from a given signal with D samples (see Fig. 1). Note that for application of this operator, it must always be ensured that the requested subsignal index i is within bounds, that is $i \in \{1, \ldots, D - d + 1\}$ must hold to address a valid subsignal.

Iterated extraction of subsignals of different length can be collapsed into one operator evaluation:

Lemma 2. Let M be a set and further let $c, d \in \mathbb{N}_1$, $c \leq d$, be two subsignal dimensionalities. Then for all $\xi \in \cup_c(M)$, $i \in \{1, \ldots, \dim_M(\xi) - d + 1\}$ and $j \in \{1, \ldots, d - c + 1\}$ it is $\text{Subsignal}_c(\text{Subsignal}_d(\xi,\ i),\ j) = \text{Subsignal}_c(\xi,\ i + j - 1)$.

Proof. The subsignal indices of the left-hand side are well within bounds. Since $i + j - 1 \in \{1, \ldots, \dim_M(\xi) - c + 1\}$ this also holds for the right-hand side. Now

$$\begin{aligned} & \text{Subsignal}_c(\text{Subsignal}_d(\xi,\ i),\ j) \\ \overset{\text{D. 1}}{=} & \sum\nolimits_{\lambda=1}^{c} \text{Subsignal}_d(\xi,\ i)_{j+\lambda-1} \cdot e_\lambda^c \\ \overset{\text{D. 1}}{=} & \sum\nolimits_{\lambda=1}^{c} \left(\sum\nolimits_{\nu=1}^{d} \xi_{i+\nu-1} \cdot e_\nu^d\right)_{j+\lambda-1} \cdot e_\lambda^c \\ \overset{(\diamondsuit)}{=} & \sum\nolimits_{\lambda=1}^{c} \xi_{(i+j-1)+\lambda-1} \cdot e_\lambda^c \\ \overset{\text{D. 1}}{=} & \text{Subsignal}_c(\xi,\ i + j - 1), \end{aligned}$$

where in the $(\diamondsuit)$ step $\nu = j + \lambda - 1$ has been substituted. □

III. Subsignal Compatible Transformations

This section introduces the concept of subsignal compatible transformations. These are functions that can be applied to an entire signal at once and then yield the same result as if they had been applied to each subsignal independently. It is shown that functions applied in a sliding fashion can be characterized as subsignal compatible transformations, and that the composition of subsignal compatible transformations is again a subsignal compatible transformation.

At the end of this section, CNNs without pooling layers are considered and it is demonstrated that these satisfy the requirements of subsignal compatible transformations. As a consequence, such networks can be applied to the whole input signal at once without having to handle individual subsignals. CNNs that *do* contain pooling layers require more theoretical preparations and are discussed verbosely in Sect. IV.

Now the primary definition of this section:

Definition 3. Let M and N be sets, let $c \in \mathbb{N}_1$ be a positive natural number, and let $T \colon \cup_c(M) \to \cup_1(N)$ be a function. T is then called a *subsignal compatible transformation with dimensionality reduction constant* c if and only if these two properties hold:

(i) *Dimensionality reduction property (DRP):* $\dim_N(T(\xi)) = \dim_M(\xi) - c + 1$ for all $\xi \in \cup_c(M)$.

(ii) *Exchange property (XP):* For all subsignal dimensionalities $d \in \mathbb{N}_1$, $d \geq c$, it holds that $T(\text{Subsignal}_d(\xi,\ i)) = \text{Subsignal}_{d-c+1}(T(\xi),\ i)$ for all $\xi \in \cup_d(M)$ and all $i \in \{1, \ldots, \dim_M(\xi) - d + 1\}$.

The first property guarantees that T reduces the dimensionality of its argument always by the same amount regardless of the concrete input. The second property states that if T is applied to an individual subsignal, then this is the same as applying T to the entire signal and afterwards extracting the appropriate samples from the resulting signal. Therefore, if with subsignal-based application of T the outcome for *all*

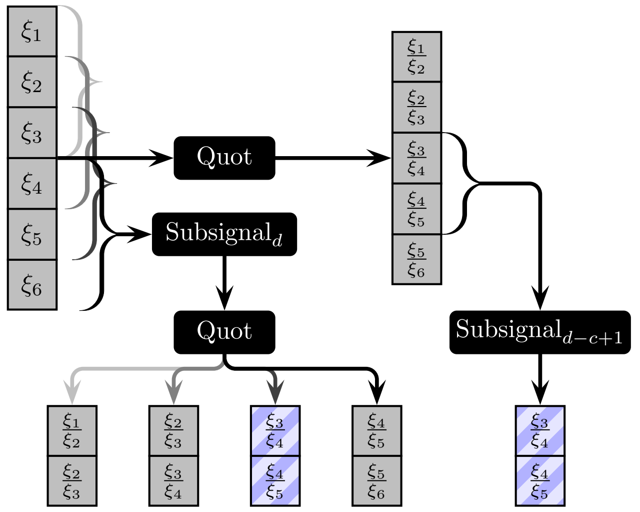


Fig. 2. Example of a subsignal compatible transformation. Here, Quot is a non-linear operator that computes the quotient of $c = 2$ adjacent samples and always reduces the dimensionality of its input by one sample, satisfying the dimensionality reduction property. The lower part shows the result of first extracting subsignals with $d = 3$ samples from the input signal ξ and then evaluating Quot. This yields processed subsignals $\text{Quot}\,(\text{Subsignal}_d(\xi,\ i))$ with $d - c + 1 = 2$ samples each. The exchange property guarantees that these processed subsignals can also be found in $\text{Quot}(\xi)$, exemplarily shown at the right-hand side of the graphics for subsignal index $i = 3$.

feasible subsignals should be determined, it suffices to carry out signal-based application of T on the entire input signal once, preventing redundant computations. These concepts are illustrated in Fig. 2.

Note that the exchange property is well-defined: The dimensionality reduction property guarantees that the dimensionalities on both sides of the equation match. Further, the subsignal index i is within bounds on both sides. This is trivial for the left-hand side, and can be seen for the right-hand side since $\dim_M(\xi) - d + 1 = (\dim_M(\xi) - c + 1) - (d - c + 1) + 1$.

An identity theorem for subsignal compatible transformations immediately follows:

Theorem 4. Let M, N be sets and $T_1, T_2\colon \cup_c(M) \to \cup_1(N)$ two subsignal compatible transformations with dimensionality reduction constant $c \in \mathbb{N}_1$. If $T_1(\rho) = T_2(\rho)$ holds for all $\rho \in M^c$, then already $T_1 = T_2$.

Proof. Let $\xi \in \cup_c(M)$. For $\mu \in \{1, \dots, \dim_M(\xi) - c + 1\}$, applying the precondition (PC) and the exchange property where the subsignal dimensionality d is set to c yields: $T_1(\xi)_\mu \overset{\text{XP}}{=} T_1(\text{Subsignal}_c(\xi,\ \mu)) \overset{\text{PC}}{=} T_2(\text{Subsignal}_c(\xi,\ \mu)) \overset{\text{XP}}{=} T_2(\xi)_\mu$. Hence all samples of the transformed signals match, thus $T_1(\xi) = T_2(\xi)$ for all ξ in the domain of T_1 and T_2. □

A. Relationship between Functions Applied in a Sliding Fashion and Subsignal Compatible Transformations

Turning now to functions applied to a signal in a *sliding* fashion, first a definition what is meant hereby:

Definition 5. Let M and N be sets, let $c \in \mathbb{N}_1$ be a positive natural number and let $f\colon M^c \to N$ be a function. Then

$$\text{Slide}_f\colon \cup_c(M) \to \cup_1(N),$$
$$\xi \mapsto \sum\nolimits_{i=1}^{\dim_M(\xi)-c+1} f(\text{Subsignal}_c(\xi,\ i)) \cdot e_i^{\dim_M(\xi)-c+1},$$

is the operator that applies f in a *sliding fashion* to all the subsignals of length c of the input signal and stores the result in a contiguous signal. The sliding window is always advanced by exactly one entry after each evaluation of f.

The next result states that functions applied in a sliding fashion are essentially the same as subsignal compatible transformations, and that the exchange property could be weakened to hold only for the case where the dimensionality reduction constant equals the subsignal dimensionality:

Theorem 6. Let M and N be sets, let $c \in \mathbb{N}_1$ and let $T\colon \cup_c(M) \to \cup_1(N)$ be a function. Then the following are equivalent:

(a) T is a subsignal compatible transformation with dimensionality reduction constant c.
(b) T fulfills the dimensionality reduction property, and for all $\xi \in \cup_c(M)$ and all $i \in \{1, \dots, \dim_M(\xi) - c + 1\}$ it holds that $T(\text{Subsignal}_c(\xi,\ i)) = T(\xi)_i$.
(c) There is a unique function $f\colon M^c \to N$ with $T = \text{Slide}_f$.

Proof. (a) ⇒ (b): Trivial, since the dimensionality reduction property is fulfilled by definition, and the claimed condition is only the special case of the exchange property where $d = c$.

(b) ⇒ (c): For showing existence, define $f\colon M^c \to N$, $\xi \mapsto T(\xi)$. For $\xi \in M^c$ it is $\dim_N(T(\xi)) = 1$ due to the dimensionality reduction property, therefore f is well-defined. Now let $\xi \in \cup_c(M)$ and define $D := \dim_M(\xi)$. It is clear that $\dim_N(T(\xi)) = \dim_N(\text{Slide}_f(\xi)) = D - c + 1$. Now let $i \in \{1, \dots, D - c + 1\}$, then the precondition (PC) implies $\text{Slide}_f(\xi)_i \overset{\text{D. 5}}{=} f(\text{Subsignal}_c(\xi,\ i)) = T(\text{Subsignal}_c(\xi,\ i)) \overset{\text{PC}}{=} T(\xi)_i$, hence $T = \text{Slide}_f$.

Considering uniqueness, suppose that there exist functions $f_1, f_2\colon M^c \to N$ with $T = \text{Slide}_{f_1} = \text{Slide}_{f_2}$. Let $\rho \in M^c$ be arbitrary, then Definition 5 gives $f_1(\rho) = \text{Slide}_{f_1}(\rho) = \text{Slide}_{f_2}(\rho) = f_2(\rho)$, therefore $f_1 = f_2$ on M^c.

(c) ⇒ (a): Suppose there is a function $f\colon M^c \to N$ with $T = \text{Slide}_f$. Slide_f inherently fulfills the dimensionality reduction property. Let $d \in \mathbb{N}_1$, $d \geq c$, be an arbitrary subsignal dimensionality and let $\xi \in \cup_d(M)$ be a signal. Further, let $i \in \{1, \dots, \dim_M(\xi) - d + 1\}$ be an arbitrary subsignal index. Remembering that $\dim_M(\text{Subsignal}_d(\xi,\ i)) = d$ and using Lemma 2 gives

$$\begin{aligned}
&\text{Slide}_f(\text{Subsignal}_d(\xi,\ i))\\
\overset{\text{D. 5}}{=}\ &\sum\nolimits_{j=1}^{d-c+1} f(\text{Subsignal}_c(\text{Subsignal}_d(\xi,\ i),\ j)) \cdot e_j^{d-c+1}\\
\overset{\text{L. 2}}{=}\ &\sum\nolimits_{j=1}^{d-c+1} f(\text{Subsignal}_c(\xi,\ i + j - 1)) \cdot e_j^{d-c+1}\\
\overset{\text{D. 5}}{=}\ &\sum\nolimits_{j=1}^{d-c+1} \text{Slide}_f(\xi)_{i+j-1} \cdot e_j^{d-c+1}\\
\overset{\text{D. 1}}{=}\ &\text{Subsignal}_{d-c+1}(\text{Slide}_f(\xi),\ i),
\end{aligned}$$

thus the exchange property is satisfied as well. □

Therefore, for each subsignal compatible transformation there is a unique function that *generates* the transformation.

This yields a succinct characterization which helps in deciding whether a given transformation fulfills the dimensionality reduction property and the exchange property. It is further clear that subsignal compatible transformation evaluations themselves can be parallelized since there is no data dependency between individual samples of the outcome.

Reconsidering Fig. 2 it is now obvious that the Quot operator introduced there is no more than the quotient of two samples evaluated in a sliding fashion. It seems plausible from this example that convolution is also a subsignal compatible transformation. This is proven rigorously in Sect. III-C.

Before discussing more theoretical properties, first an example of a transformation that is *not* subsignal compatible:

Example 7. Let $\mathbb{Z}$ denote the integers and consider the function $T\colon \cup_1(\mathbb{Z}) \to \cup_1(\mathbb{Z})$, $\xi \mapsto (-1)^{\dim_{\mathbb{Z}}(\xi)} \cdot \xi$, which fulfills the dimensionality reduction property with dimensionality reduction constant $c := 1$. The exchange property is, however, not satisfied: Let $d := 1$ and $\xi \in \mathbb{Z}^2$, then $i := 1$ yields $T(\text{Subsignal}_d(\xi,\ i)) = T(\xi_1) = -\xi_1$, but it is $\text{Subsignal}_{d-c+1}(T(\xi),\ i) = \text{Subsignal}_{d-c+1}(\xi) = \xi_1$. Since $\xi_1 \neq -\xi_1$ unless ξ_1 vanishes, T cannot be a subsignal compatible transformation.

B. Composition of Subsignal Compatible Transformations

The composition of subsignal compatible transformations is again a subsignal compatible transformation, where the dimensionality reduction constant has to be adjusted:

Theorem 8. Let M, N and P be sets and let $c_1, c_2 \in \mathbb{N}_1$. Suppose $T_1\colon \cup_{c_1}(M) \to \cup_1(N)$ is a subsignal compatible transformation with dimensionality reduction constant c_1, and $T_2\colon \cup_{c_2}(N) \to \cup_1(P)$ is a subsignal compatible transformation with dimensionality reduction constant c_2.

Define $c := c_1 + c_2 - 1 \in \mathbb{N}_1$. Then $T\colon \cup_c(M) \to \cup_1(P)$, $\xi \mapsto T_2(T_1(\xi))$, is a subsignal compatible transformation with dimensionality reduction constant c.

Proof. Note first that $c \geq 1$ since $c_1 \geq 1$ and $c_2 \geq 1$, hence indeed $c \in \mathbb{N}_1$. Let $\xi \in \cup_c(M)$ be arbitrary for demonstrating that T is well-defined. As $c \geq c_1$ because of $c_2 \geq 1$, this yields $\cup_c(M) \subseteq \cup_{c_1}(M)$ and hence $T_1(\xi)$ is well-defined. Further, $\dim_N(T_1(\xi)) = \dim_M(\xi) - c_1 + 1 \geq c - c_1 + 1 = c_2$ using the dimensionality reduction property of T_1, therefore $T_1(\xi) \in \cup_{c_2}(N)$. Thus $T_2(T_1(\xi))$ is well-defined, and so is T.

For all $\xi \in \cup_c(M)$, the dimensionality reduction property of T_1 and T_2 now implies $\dim_P(T(\xi)) = \dim_P(T_2(T_1(\xi))) \overset{\text{DRP}}{=} \dim_N(T_1(\xi)) - c_2 + 1 \overset{\text{DRP}}{=} \dim_M(\xi) - c_1 + 1 - c_2 + 1 = \dim_M(\xi) - c + 1$, therefore T fulfills the dimensionality reduction property.

Let $d \in \mathbb{N}_1$, $d \geq c$, be arbitrary, and let $\xi \in \cup_d(M)$ and $i \in \{1, \dots, \dim_M(\xi) - d + 1\}$. Since both T_1 and T_2 satisfy the exchange property, it follows that $T(\text{Subsignal}_d(\xi,\ i)) \overset{\text{XP}}{=} T_2(\text{Subsignal}_{d-c_1+1}(T_1(\xi,\ i))) \overset{\text{XP}}{=} \text{Subsignal}_{d-c+1}(T(\xi),\ i)$, where $d \geq c_1$ and $d - c_1 + 1 \geq c_2$ hold during the two respective applications of the exchange property. Therefore, T also fulfills the exchange property. □

This result can be generalized immediately to compositions of more than two subsignal compatible transformations:

Corollary 9. Let $n \in \mathbb{N}$, $n \geq 2$, and let $M_1, \dots, M_{n+1}$ be sets. For each $\lambda \in \{1, \dots, n\}$ let $T_\lambda\colon \cup_{c_\lambda}(M_\lambda) \to \cup_1(M_{\lambda+1})$ be a subsignal compatible transformation with dimensionality reduction constant $c_\lambda \in \mathbb{N}_1$. Then the composed function $T\colon \cup_c(M_1) \to \cup_1(M_{n+1})$, $\xi \mapsto \left(\circ_1^{\lambda=n} T_\lambda\right)(\xi)$, is a subsignal compatible transformation with dimensionality reduction constant $c := \sum_{\mu=1}^{n} c_\mu - n + 1 \in \mathbb{N}_1$.

Proof. Define $S_1 := T_1$, and for each $\lambda \in \{2, \dots, n\}$ let $S_\lambda\colon \cup_{\sum_{\mu=1}^{\lambda} c_\mu - \lambda + 1}(M_1) \to \cup_1(M_{\lambda+1})$, $\xi \mapsto T_\lambda(S_{\lambda-1}(\xi))$, be a function. Since $T = S_n$, the claim follows when it is shown with induction for λ that S_λ is a subsignal compatible transformation with dimensionality reduction constant $\sum_{\mu=1}^{\lambda} c_\mu - \lambda + 1$. While the situation $\lambda = 1$ is trivial, the induction step follows with Theorem 8. □

C. CNNs without Pooling Layers

To conclude this section, a demonstration is provided of how CNNs without any pooling layers fit in the theory developed so far. Since pooling layers require a non-trivial extension of the theory, they are detailed in Sect. IV.

Convolutional layers are the most substantial ingredient of CNNs, the trainable degrees of freedom which facilitate adaptation of the network to a specific task are located here. In these layers, multi-channel input feature maps are convolved channel-wise with adjustable filter banks, the result is accumulated and an adjustable bias is added to yield the output feature map.

First, the introduction of the indexing rules for iterated structures to account for the multi-channel nature of the occurring signals. Let M be a set, $a, b \in \mathbb{N}_1$ positive natural numbers and $\xi \in (M^a)^b$ a multi-channel signal. It is then $\xi_j \in M^a$ for indices $j \in \{1, \dots, b\}$, and moreover $(\xi_j)_i \in M$ for indices $j \in \{1, \dots, b\}$ and $i \in \{1, \dots, a\}$. This rule is extended naturally to sets written explicitly as products with more than two factors. Therefore, if $\xi \in ((M^a)^b)^c$ for another number $c \in \mathbb{N}_1$, then for example $(\xi_k)_j \in M^a$ for indices $k \in \{1, \dots, c\}$ and $j \in \{1, \dots, b\}$.

These rules become clearer if the multi-channel convolution operation $*$ is considered. Suppose the samples are members of a ring R, $m \in \mathbb{N}_1$ denotes the number of input channels, $n \in \mathbb{N}_1$ is the number of output channels, and $c \in \mathbb{N}_1$ equals the number of samples considered at any one time during convolution with the filter bank, or in other words the receptive field size of the convolutional layer. Then input signals or feature maps with $D \in \mathbb{N}_1$ samples have form $\xi \in (R^m)^D$, and filter banks can be represented by a tensor $w \in ((R^n)^m)^c$. Here $D \geq c$ must hold, that is the filter kernel should be smaller than the input signal.

The output feature map $(\xi * w) \in (R^n)^{D-c+1}$ is then

$$(\xi * w)_i := \sum\nolimits_{\lambda=1}^{m} \sum\nolimits_{\mu=1}^{c} (w_\mu)_\lambda \cdot (\xi_{c+i-\mu})_\lambda \in R^n$$

for indices $i \in \{1, \dots, D-c+1\}$. Note that $(w_\mu)_\lambda \in R^n$ and $(\xi_{c+i-\mu})_\lambda \in R$, so that the result of their product is understood here as scalar product. The operation is well-defined since

$c + i - \mu \in \{1, \dots, D\}$, which follows immediately through substitution of the extreme values of i and μ.

This multi-channel convolution operation is indeed a subsignal compatible transformation as shown explicitly here:

Example 10. Define $M := R^m$ and $N := R^n$ and consider $f_{\text{conv}}\colon M^c \to N$,

$$\xi \mapsto \sum\nolimits_{\lambda=1}^{m} \sum\nolimits_{\mu=1}^{c} (w_\mu)_\lambda \cdot (\xi_{c-\mu+1})_\lambda.$$

Since $\mu \in \{1, \dots, c\}$ it is $c - \mu + 1 \in \{1, \dots, c\}$, hence f_{conv} is well-defined. For all $\xi \in M^D$ and any $i \in \{1, \dots, D - c + 1\}$ follows

$$\begin{aligned} & \text{Slide}_{f_{\text{conv}}}(\xi)_i \\ \overset{\text{D. 5}}{=} \; & f_{\text{conv}}(\text{Subsignal}_c(\xi,\ i)) \\ \overset{\text{D. 1}}{=} \; & f_{\text{conv}}\left(\sum\nolimits_{\nu=1}^{c} \xi_{i+\nu-1} \cdot e_\nu^c\right) \\ = \; & \sum\nolimits_{\lambda=1}^{m} \sum\nolimits_{\mu=1}^{c} (w_\mu)_\lambda \cdot \left(\left(\sum\nolimits_{\nu=1}^{c} \xi_{i+\nu-1} \cdot e_\nu^c\right)_{c-\mu+1}\right)_\lambda \\ \overset{(\Diamond)}{=} \; & \sum\nolimits_{\lambda=1}^{m} \sum\nolimits_{\mu=1}^{c} (w_\mu)_\lambda \cdot (\xi_{i+c-\mu+1-1})_\lambda \\ = \; & (\xi * w)_i, \end{aligned}$$

where $\nu = c - \mu + 1$ was substituted in the $(\Diamond)$ step. The multi-channel convolution operation as defined above is hence in fact the application of f_{conv} in a sliding fashion. Therefore, Theorem 6 guarantees that $*$ is a subsignal compatible transformation with dimensionality reduction constant c.

Since fully-connected layers are merely a special case of convolutional layers, these do not need any special treatment here. Addition of biases does not require any knowledge on the spatial structure of the convolution's result and is therefore a trivial subsignal compatible transformation with dimensionality reduction constant 1. Non-linearity layers are nothing but the application of a scalar-valued function to all the samples of an input signal. Hence these layers also form subsignal compatible transformations with dimensionality reduction constant 1 due to Theorem 6.

Furthermore, compositions of these operations can also be understood as subsignal compatible transformations with Corollary 9. As a consequence, the exchange property facilitates application of CNNs without pooling layers to an entire signal at once instead of each subsignal independently, all without incurring any accuracy loss. The next section will extend this result to CNNs that may also feature pooling layers.

IV. Pooling Layers and Functions Applied in a Strided Fashion

So far it has been shown how convolutional layers and non-linearity layers of a CNN fit in the theoretical framework of subsignal compatible transformations. This section analyzes pooling layers which apply a pooling kernel to non-overlapping blocks of the input signal. This is equivalent to a function applied in a sliding fashion followed by a downsampling operation, which will here be referred to as the application of a function in a *strided* fashion.

The theory developed herein can of course also be applied to other functions than the pooling kernels encountered in ordinary CNNs. For example, multi-channel convolution in which the filter bank is advanced by the receptive field size is essentially f_{conv} from Sect. III-C applied in a strided fashion. Application of convolution where the filter banks are advanced by more than one sample has however no benefit in terms of execution speed for signal-based application. This is discussed at the end of Sect. IV-B after having developed sufficient theory to analyze this notion.

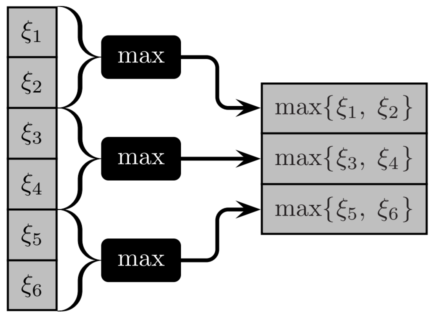


Fig. 3. Illustration of the pooling kernel max that determines the maximum of $k = 2$ adjacent samples. It is here applied in a strided fashion to non-overlapping subsignals of an input signal ξ with $D = 6$ samples, yielding an output signal $\text{Stride}_{\max}(\xi)$ with $\frac{D}{k} = 3$ samples. Since here the dimensionality halves and therefore the dimensionality reduction property is violated, this is *not* a subsignal compatible transformation.

This section demonstrates how these functions can be turned into efficiently computable subsignal compatible transformations using a data structure recently introduced as fragmentation by Giusti *et al.* [14]. Here, that proposed method is generalized and rigorously proven correct. As an added benefit of these results, the dynamics of the entire signal processing chain can also be accurately described, including the possibility of tracking down the position of each processed subsignal in the fragmentation data structure.

Moreover, the circumstances under which the fragment dimensionalities are guaranteed to always be homogeneous are analyzed. This is a desirable property as it facilitates the application of subsequent operations to signals which all have the same number of samples, rendering cumbersome handling of special cases obsolete and thus resulting in accelerated execution on massively parallel processors. For CNNs this means that conventional tensor convolutions can be used without any modifications whatsoever, which is especially beneficial if a highly-optimized implementation is readily available.

First, a more precise statement on what the application of a function in a strided fashion means (see Fig. 3 for orientation):

Definition 11. Let M and N be sets, let $k \in \mathbb{N}_1$ be a positive natural number and let $g\colon M^k \to N$ be a function. Then $\text{Stride}_g\colon \cup_{q=1}^{\infty} M^{kq} \to \cup_1(N)$,

$$\xi \mapsto \sum\nolimits_{i=1}^{\dim_M(\xi)/k} g(\text{Subsignal}_k(\xi,\ k(i-1)+1)) \cdot e_i^{\dim_M(\xi)/k},$$

is the operator that applies g in a *strided fashion* to signals where the number of samples is a multiple of k. The subsignal indices are chosen here so that all non-overlapping subsignals are fed through g, starting with the first valid subsignal.

Since it is $k(i-1)+1 \in \{1, \dots, \dim_M(\xi) - k + 1\}$ for all $i \in \{1, \dots, \dim_M(\xi)/k\}$, Stride_g is well-defined. Further,

$\dim_M(\xi)/\dim_N(\mathrm{Stride}_g(\xi)) = k$ for all ξ in the domain of Stride_g. Since the input dimensionality is reduced here through division with a natural number rather than a subtraction, the dimensionality reduction property cannot be fulfilled unless $k = 1$. The situation in which $k = 1$ is, however, not particularly interesting since then $\mathrm{Stride}_g = \mathrm{Slide}_g$ which was already handled in Sect. III.

Before continuing with fragmentation, first consider multichannel pooling kernels commonly encountered in CNNs:

Example 12. Assume the goal is to process real-valued signals with $m \in \mathbb{N}_1$ channels, that is $M = N = \mathbb{R}^m$, where each channel should be processed independently of the others, and $k \in \mathbb{N}_1$ adjacent samples should be compressed into one output sample. *Average pooling* is then realized by the pooling kernel $g_{\mathrm{avg}}(\xi) := \frac{1}{k}\sum_{\nu=1}^{k} \xi_\nu$, which determines the channel-wise empirical mean value of the samples. Another example is *max-pooling*, where the maximum entry in each channel should be determined. This can be achieved with the pooling kernel $g_{\max}(\xi) := \sum_{\lambda=1}^{m} \left(\max_{\nu=1}^{k}(\xi_\nu)_\lambda\right) \cdot e_\lambda^m$.

A. Fragmentation

The fragmentation operator [14] performs a spatial reordering operation. Its precise analysis requires a recap of some elementary number theory. For all numbers $a \in \mathbb{N}$ and $b \in \mathbb{N}_1$, *Euclidean division* guarantees that there are unique numbers $\mathrm{div}(a,\ b) \in \mathbb{N}$ and $\mathrm{rem}(a,\ b) \in \{0, \dots, b-1\}$ so that $a = \mathrm{div}(a,\ b) \cdot b + \mathrm{rem}(a,\ b)$. Here is a small collection of results on these operators for further reference:

Proposition 13. It is $\mathrm{div}(a,\ 1) = a$ and $\mathrm{rem}(a,\ 1) = 0$ for all $a \in \mathbb{N}$. Moreover, $\mathrm{div}(a + bc,\ c) = \mathrm{div}(a,\ c) + b$ and $\mathrm{rem}(a + bc,\ c) = \mathrm{rem}(a,\ c)$ for all $a, b \in \mathbb{N}$ and $c \in \mathbb{N}_1$.

If the fragmentation operator is applied to a signal, it puts certain samples into individual fragments which can be grasped as signals themselves. If a collection of fragments is fragmented further, a larger collection of fragments results. The total number of samples is, however, left unchanged after these operations. For the sake of convenience, matrices are used here as concrete data structure for fragmented signals, where columns correspond to fragments and rows correspond to signal samples.

First, some notation needs to be defined. If M is a set and $a, b \in \mathbb{N}_1$, then $M^{a \times b}$ denotes the set of all matrices with a rows and b columns with entries from M. In the present context, this represents a collection of b fragments where each signal has a samples. For $\xi \in M^{a\times b}$, $\mathrm{rdim}_M(\xi) = a$ and $\mathrm{cdim}_M(\xi) = b$ denote the number of rows and columns, respectively. Furthermore, $\xi_{i,\ j}$ is the entry in the i-th row and j-th column of ξ where $i \in \{1, \dots, a\}$ and $j \in \{1, \dots, b\}$. The transpose of ξ is written as ξ^T.

The vectorization operator [21] stacks all the columns of a matrix on top of another:

Definition 14. Let M be a set and $a, b \in \mathbb{N}_1$. The *vectorization operator* $\mathrm{vec}_{a\times b}\colon M^{a\times b} \to M^{ab}$ is characterized by $\mathrm{vec}_{a\times b}(\xi)_j = \xi_{\mathrm{rem}(j-1,\ a)+1,\ \mathrm{div}(j-1,\ a)+1}$ for all indices $j \in \{1, \dots, ab\}$ and all matrices $\xi \in M^{a\times b}$. The *inverse vectorization operator* $\mathrm{vec}_{a\times b}^{-1}\colon M^{ab} \to M^{a\times b}$ is given by $\mathrm{vec}_{a\times b}^{-1}(\xi)_{i,\ j} = \xi_{(j-1)a+i}$ for all indices $i \in \{1, \dots, a\}$, $j \in \{1, \dots, b\}$ and all vectors $\xi \in M^{ab}$.

It can be verified directly that these two operators are well-defined permutations and inversely related to one another. With their help the fragmentation operator may now be defined:

Definition 15. Let M be a set and $k \in \mathbb{N}_1$. For arbitrary vector dimensionalities $q \in \mathbb{N}_1$ and numbers of input fragments $s \in \mathbb{N}_1$ the function $\mathrm{Frag}_k\colon M^{kq\times s} \to M^{q\times ks}$,

$$\xi \mapsto \left(\mathrm{vec}_{ks\times q}^{-1}\left(\mathrm{vec}_{s\times kq}\left(\xi^T\right)\right)\right)^T,$$

is called the *fragmentation operator*.

Here, k equals the corresponding parameter from the application of a function in a strided fashion. Frag_k is clearly well-defined, and the number of output fragments is ks. Next consider this operator that undoes the ordering of the fragmentation operator:

Definition 16. Let M be a set, let $k \in \mathbb{N}_1$, and let $q \in \mathbb{N}_1$ denote a vector dimensionality and $s \in \mathbb{N}_1$ a number of output fragments. Then $\mathrm{Defrag}_k\colon M^{q\times ks} \to M^{kq\times s}$,

$$\xi \mapsto \left(\mathrm{vec}_{s\times kq}^{-1}\left(\mathrm{vec}_{ks\times q}\left(\xi^T\right)\right)\right)^T,$$

is called the *defragmentation operator*.

Note that Defrag_k is well-defined and the number of input fragments must equal ks. Fragmentation and defragmentation are inversely related, that is $\mathrm{Defrag}_k \circ \mathrm{Frag}_k = \mathrm{id}_{M^{kq\times s}}$ and $\mathrm{Frag}_k \circ \mathrm{Defrag}_k = \mathrm{id}_{M^{q\times ks}}$. An illustration of the operations performed during fragmentation and defragmentation is depicted in Fig. 4.

Fragmentation is merely a certain reordering operation:

Lemma 17. Suppose that M is a set, $k, q, s \in \mathbb{N}_1$ and $\xi \in M^{kq\times s}$. Then $\mathrm{rdim}_M(\mathrm{Frag}_k(\xi)) = \frac{1}{k} \cdot \mathrm{rdim}_M(\xi)$ and $\mathrm{cdim}_M(\mathrm{Frag}_k(\xi)) = k \cdot \mathrm{cdim}_M(\xi)$. Further, $\mathrm{Frag}_k(\xi)_{\mu,\ \nu} = \xi_{\mathrm{div}((\mu-1)ks+\nu-1,\ s)+1,\ \mathrm{rem}((\mu-1)ks+\nu-1,\ s)+1}$ for all indices $\mu \in \{1, \dots, q\}$ and $\nu \in \{1, \dots, ks\}$.

Proof. The dimensionality statements are obvious by the definition of Frag_k. To prove the identity, let $\mu \in \{1, \dots, q\}$ and $\nu \in \{1, \dots, ks\}$. One yields

$$\begin{aligned} &\mathrm{Frag}_k(\xi)_{\mu,\ \nu} \\ \overset{\text{D. 15}}{=}\ &\mathrm{vec}_{ks\times q}^{-1}\left(\mathrm{vec}_{s\times kq}\left(\xi^T\right)\right)_{\nu,\ \mu} \overset{\text{D. 14}}{=} \mathrm{vec}_{s\times kq}\left(\xi^T\right)_{(\mu-1)ks+\nu} \\ \overset{\text{D. 14}}{=}\ &\left(\xi^T\right)_{\mathrm{rem}((\mu-1)ks+\nu-1,\ s)+1,\ \mathrm{div}((\mu-1)ks+\nu-1,\ s)+1}, \end{aligned}$$

and the claim follows. □

Similar properties are fulfilled by defragmentation:

Lemma 18. Let M be a set. Let $k, q, s \in \mathbb{N}_1$ be positive natural numbers and let $\xi \in M^{q\times ks}$ be an arbitrary fragmented signal. Then $\mathrm{rdim}_M(\mathrm{Defrag}_k(\xi)) = k \cdot \mathrm{rdim}_M(\xi)$, $\mathrm{cdim}_M(\mathrm{Defrag}_k(\xi)) = \frac{1}{k} \cdot \mathrm{cdim}_M(\xi)$, and $\mathrm{Defrag}_k(\xi)_{\mu,\ \nu} = \xi_{\mathrm{div}((\mu-1)s+\nu-1,\ ks)+1,\ \mathrm{rem}((\mu-1)s+\nu-1,\ ks)+1}$ for all indices $\mu \in \{1, \dots, kq\}$, $\nu \in \{1, \dots, s\}$.

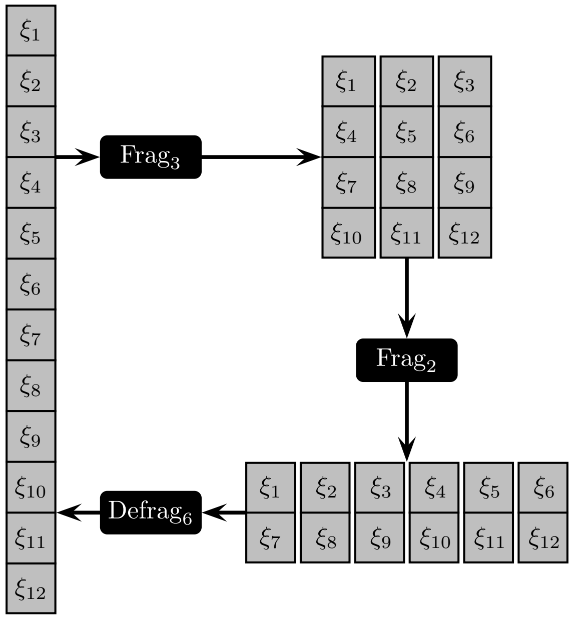


Fig. 4. Illustration of the fragmentation and defragmentation operators. The left-hand side shows an input signal ξ with $q_0 = 12$ samples in a single fragment, that is $s_0 = 1$. Fragmentation with the parameter $k_1 = 3$ yields a signal with $q_1 = \frac{q_0}{k_1} = 4$ samples in each of $s_1 = k_1 s_0 = 3$ fragments, shown at the upper right in the graphics. A second application of fragmentation with $k_2 = 2$ results in $s_2 = k_2 s_1 = 6$ fragments with $q_2 = \frac{q_1}{k_2} = 2$ samples each, depicted at the lower right. A complete defragmentation with parameter $k^* = k_1 k_2 = 6$ yields the original input signal ξ again, with $q_3 = k^* q_2 = 12$ samples in a single fragment, that is $s_3 = \frac{s_2}{k^*} = 1$.

Proof. Completely analogous to Lemma 17. □

As already outlined in Fig. 4, compositions of the fragmentation operator are equivalent to a single fragmentation with an adjusted parameterization:

Remark 19. Let M be a set and $k_1, k_2, q, s \in \mathbb{N}_1$. Then $\mathrm{Frag}_{k_2}(\mathrm{Frag}_{k_1}(\xi)) = \mathrm{Frag}_{k_1 k_2}(\xi)$ for all $\xi \in M^{k_1 k_2 q \times s}$.

Proof. The claim follows through entry-wise comparison between $\mathrm{Frag}_{k_2}(\mathrm{Frag}_{k_1}(\xi))$ and $\mathrm{Frag}_{k_1 k_2}(\xi)$ using Lemma 17. □

It follows immediately that fragmentation is a commutative operation:

Remark 20. If M denotes a set, $k_1, k_2, q, s \in \mathbb{N}_1$ are natural numbers and $\xi \in M^{k_1 k_2 q \times s}$ is a fragmented signal, then $\mathrm{Frag}_{k_2}(\mathrm{Frag}_{k_1}(\xi)) = \mathrm{Frag}_{k_1}(\mathrm{Frag}_{k_2}(\xi))$.

Proof. Obvious with Remark 19 as multiplication in $\mathbb{N}_1$ is commutative. □

B. Relationship between Fragmentation, Functions Applied in a Strided Fashion and Subsignal Compatible Transformations

A bit more background is necessary before analyzing how functions applied in a strided fashion fit into the theory of subsignal compatible transformations. The outcome of a subsignal compatible transformation applied to a fragmented signal is defined naturally:

Definition 21. Let M, N be sets and $T\colon \cup_c(M) \to \cup_1(N)$ a subsignal compatible transformation with dimensionality reduction constant $c \in \mathbb{N}_1$. Let $\xi \in M^{D \times s}$ be a fragmented signal with $D \in \mathbb{N}_1$ samples in each of the $s \in \mathbb{N}_1$ fragments, where $D \geq c$ holds. For $\gamma \in \{1, \dots, s\}$ let $\xi^{(\gamma)} := \sum_{\nu=1}^{D} \xi_{\nu,\gamma} \cdot e_\nu^D \in M^D$ denote the individual fragments. The output of T applied to ξ is then defined as $T(\xi) := [T(\xi^{(1)}), \dots, T(\xi^{(s)})] \in N^{(D-c+1) \times s}$, that is T is applied to all the fragments independently.

Since there is no data dependency between fragments, parallelization of subsignal compatible transformation evaluation over all output samples is straightforward. What follows is the formal introduction of the processing chain concept which captures and generalizes all the dynamics of a CNN, and two notions of its application to signal processing:

Definition 22. The collection of the following objects is called a *processing chain*: A fixed subsignal dimensionality $B \in \mathbb{N}_1$, a number of layers $L \in \mathbb{N}_1$, a sequence of sets $M_0, \dots, M_L, N_1, \dots, N_L$, and for each $j \in \{1, \dots, L\}$ subsignal compatible transformations $T_j\colon \cup_{c_j}(M_{j-1}) \to \cup_1(N_j)$ with dimensionality reduction constant $c_j \in \mathbb{N}_1$ and functions $g_j\colon N_j^{k_j} \to M_j$ where $k_j \in \mathbb{N}_1$. The numbers $k_j^* := \prod_{\nu=1}^{j} k_\nu$ for $j \in \{0, \dots, L\}$ are called the *stride products* of the processing chain. This implies that $k_0^* = 1$. For $j \in \{0, \dots, L\}$, the operator $\mathrm{EvalStride}_j\colon M_0^B \to \cup_1(M_j)$,

$$\rho \mapsto \begin{cases} \rho, & \text{if } j = 0, \\ \mathrm{Stride}_{g_j}(T_j(\mathrm{EvalStride}_{j-1}(\rho))), & \text{if } j > 0, \end{cases}$$

applies the processing chain in a *strided* fashion, and further $\mathrm{EvalSlide}_j\colon \cup_B(M_0) \to \cup_{q=1}^{\infty} \cup_{s=1}^{\infty} M_j^{q \times s}$,

$$\xi \mapsto \begin{cases} \xi, & \text{if } j = 0, \\ \mathrm{Frag}_{k_j}(\mathrm{Slide}_{g_j}(T_j(\mathrm{EvalSlide}_{j-1}(\xi)))), & \text{if } j > 0, \end{cases}$$

is the operator that applies the processing chain in a *sliding* fashion. Note that these two functions are *not* well-defined unless additional conditions are fulfilled, detailed below.

The number B here represents the extent of the region that is fed into a CNN, or in other words the entire network's receptive field size. This size is a design parameter of the network and depends on the concrete definitions of all of its layers. The functions T_j in a processing chain can be substituted with the appropriate layer types discussed earlier, such as convolutions or non-linearities, or compositions thereof. Pooling kernels and other functions applied in a strided fashion to non-overlapping blocks can be plugged into a processing chain via the g_j functions. The recursive definitions of $\mathrm{EvalStride}_j$ and $\mathrm{EvalSlide}_j$ represent the alternating evaluation of a subsignal compatible transformation and a function applied in a strided fashion up to the specified layer index j.

The rationale for the $\mathrm{EvalStride}$ operator is the naive *subsignal-based application* of a CNN: Here, the CNN is applied in the ordinary way to signals of length equal to the network's receptive field size B. According application of the network using a sliding window approach involves extraction of all feasible overlapping subsignals of length B and feeding them through the network independently of each other.

The $\mathrm{EvalSlide}$ operator differs from $\mathrm{EvalStride}$ in that it corresponds to the *signal-based application* of a CNN:

No overlapping subsignals need to be processed separately here, preventing redundant computations. Instead, the complete input signal is processed in its entirety, sharing intermediate computation results among adjacent subsignals. Using $\mathrm{EvalSlide}$, the g_j functions are applied in a sliding rather than a strided fashion, followed by a fragmentation operation.

Definition 22 thus describes a recipe for how a CNN can be transformed from a subsignal-based application to signal-based application. A concrete example will be discussed in Sect. VI. First, however, a theoretical justification that this method indeed produces the correct outcome under all circumstances will be presented. The next result states when the application of a processing chain is well-defined, and it proves that the result of the $\mathrm{EvalStride}$ operator applied to a subsignal of a larger signal can be found within the result of $\mathrm{EvalSlide}$ applied to the entire signal. This then implies that both approaches deliver the very same values and hence verify $\mathrm{EvalSlide}$ involves no accuracy loss whatsoever.

Lemma 23. Given a processing chain with the same notation as in Definition 22, first assume that k_j divides $\dim_{N_j}(T_j(\mathrm{EvalStride}_{j-1}(\rho)))$ and that $\mathrm{EvalStride}_j(\rho)$ is non-empty for all $j \in \{1, \dots, L\}$ and all $\rho \in M_0^B$. In other words, the application of the processing chain in a strided fashion should be well-defined.

Let $D \in \mathbb{N}_1$, $D \geq B$, be a signal dimensionality so that the number of subsignals $D - B + 1$ of length B is divisible by the final stride product k_L^*, and let $\xi \in M_0^D$ be the considered signal. Then the application of the processing chain in a sliding fashion to ξ is well-defined, and additional statements hold:

Let $u_j := \dim_{M_j}(\mathrm{EvalStride}_j(\mathrm{Subsignal}_B(\xi,\ i))) \in \mathbb{N}_1$ for all $j \in \{0, \dots, L\}$ be an abbreviation for the dimensionality of the intermediate representations in each layer of the $\mathrm{EvalStride}$ cascade. Note that these numbers are actually independent of any subsignal index i. Further, for all $j \in \{0, \dots, L\}$, let $U_j^{\mathrm{col}} := \mathrm{cdim}_{M_j}(\mathrm{EvalSlide}_j(\xi)) \in \mathbb{N}_1$ and $U_j^{\mathrm{row}} := \mathrm{rdim}_{M_j}(\mathrm{EvalSlide}_j(\xi)) \in \mathbb{N}_1$ be defined as abbreviations for the number of fragments and the fragmented signal dimensionality, respectively, after each layer using the $\mathrm{EvalSlide}$ operator. Then for all $j \in \{0, \dots, L\}$ the following holds:

(a) $u_j = \frac{1}{k_j^*}\left(B - \sum_{\mu=1}^{j} k_{\mu-1}^*(c_\mu - 1)\right)$.

(b) $U_j^{\mathrm{row}} = \frac{1}{k_j^*}\left(D - k_j^* + 1 - \sum_{\mu=1}^{j} k_{\mu-1}^*(c_\mu - 1)\right)$ and $U_j^{\mathrm{col}} = k_j^*$.

(c) $U_j^{\mathrm{row}} - u_j + 1 = \frac{1}{k_j^*}(D - B + 1)$. In other words, the number of distinct subsignals with u_j samples in each fragment of the fragmented signals equals the original number of distinct subsignals divided by the corresponding number of fragments.

(d) For $i \in \{1, \dots, D - B + 1\}$ and $\mu \in \{1, \dots, u_j\}$ it is

$$\begin{aligned}&\mathrm{EvalStride}_j(\mathrm{Subsignal}_B(\xi,\ i))_\mu \\ =\ &\mathrm{EvalSlide}_j(\xi)_{\mathrm{div}(i-1,\ k_j^*)+\mu,\ \mathrm{rem}(i-1,\ k_j^*)+1}.\end{aligned}$$

Here, the latter can also be understood as one sample of the $\mathrm{Subsignal}_{u_j}$ operator applied to a certain fragment of $\mathrm{EvalSlide}_j(\xi)$.

Proof. (a) Let $i \in \{1, \dots, D - B + 1\}$ be arbitrary and define $\rho := \mathrm{Subsignal}_B(\xi,\ i) \in M_0^B$ as an abbreviation. It is $u_0 = \dim_{M_0}(\mathrm{EvalStride}_0(\rho)) = \dim_{M_0}(\rho) = B$, and the right-hand side of the claim trivially equals B for $j = 0$. Carrying out induction for $j - 1 \to j$ yields:

$$\begin{aligned}
u_j &\overset{\text{D. 22}}{=} \dim_{M_j}(\mathrm{Stride}_{g_j}(T_j(\mathrm{EvalStride}_{j-1}(\rho)))) \\
&\overset{\text{D. 11}}{=} \tfrac{1}{k_j}\dim_{N_j}(T_j(\mathrm{EvalStride}_{j-1}(\rho))) \\
&\overset{\text{DRP}}{=} \tfrac{1}{k_j}\left(\dim_{M_{j-1}}(\mathrm{EvalStride}_{j-1}(\rho)) - c_j + 1\right) \\
&\overset{\text{IH}}{=} \tfrac{1}{k_j}\left(\tfrac{1}{k_{j-1}^*}\left(B - \sum_{\mu=1}^{j-1} k_{\mu-1}^*(c_\mu - 1)\right) - (c_j - 1)\right) \\
&= \tfrac{1}{k_j k_{j-1}^*}\left(B - \sum_{\mu=1}^{j-1} k_{\mu-1}^*(c_\mu - 1) - k_{j-1}^*(c_j - 1)\right),
\end{aligned}$$

where IH denotes substitution of the induction hypothesis. Hence, the claimed expression follows since $k_j^* = k_j k_{j-1}^*$. Note that u_j is indeed a positive natural number because k_j divides $\dim_{N_j}(T_j(\mathrm{EvalStride}_{j-1}(\rho)))$ and $\mathrm{EvalStride}_j(\rho)$ is non-empty by requirement.

(b) Besides the statements on U_j^{row} and U_j^{col} it is shown here that the application of the processing chain in a sliding fashion is well-defined using induction for j. For $j = 0$ follows $\mathrm{EvalSlide}_0(\xi) = \xi$, which is trivially well-defined and by definition it is $\xi \in M_0^D = M_0^{D \times 1}$. Therefore, $U_0^{\mathrm{row}} = D$ and $U_0^{\mathrm{col}} = 1$ which equals the claimed expressions since $k_0^* = 1$.

For $j - 1 \to j$, it is first demonstrated that k_j divides $\chi := \mathrm{rdim}_{M_j}(\mathrm{Slide}_{g_j}(T_j(\mathrm{EvalSlide}_{j-1}(\xi))))$ which implies well-definedness since the fragmentation operator can then indeed be applied. It follows that

$$\begin{aligned}
\chi &\overset{\text{D. 5}}{=} \mathrm{rdim}_{N_j}(T_j(\mathrm{EvalSlide}_{j-1}(\xi))) - k_j + 1 \\
&\overset{\text{DRP}}{=} \mathrm{rdim}_{M_{j-1}}(\mathrm{EvalSlide}_{j-1}(\xi)) - c_j + 1 - k_j + 1 \\
&= U_{j-1}^{\mathrm{row}} - c_j + 1 - k_j + 1 \\
&\overset{\text{IH}}{=} \tfrac{1}{k_{j-1}^*}\left(D - k_{j-1}^* + 1 - \sum_{\mu=1}^{j-1} k_{\mu-1}^*(c_\mu - 1)\right) \\
&\quad + \tfrac{1}{k_{j-1}^*}\left(-k_{j-1}^*(c_j - 1) - k_j^* + k_{j-1}^*\right) \\
&= \tfrac{1}{k_{j-1}^*}\left(D - k_j^* + 1 - \sum_{\mu=1}^{j} k_{\mu-1}^*(c_\mu - 1)\right).
\end{aligned}$$

By requirement on the signal length D there exists a number $t \in \mathbb{N}_1$ so that $D - B + 1 = k_L^* t$. Substitution yields

$$\begin{aligned}
\chi &= \tfrac{1}{k_{j-1}^*}\left(B - \sum_{\mu=1}^{j} k_{\mu-1}^*(c_\mu - 1) + k_L^* t - k_j^*\right) \\
&\overset{\text{(a)}}{=} k_j u_j + k_j \cdots k_L \cdot t - k_j.
\end{aligned}$$

Proposition 13 implies that k_j divides χ since $u_j \in \mathbb{N}_1$ as shown in (a), hence the processing chain can be applied until the j-th layer. With Lemma 17 follows $U_j^{\mathrm{row}} = \frac{1}{k_j}\chi$, which immediately yields the claimed expression. Since only fragmentation influences the number of columns in the processing chain application, it follows that $U_j^{\mathrm{col}} = k_j U_{j-1}^{\mathrm{col}}$ with Lemma 17, proving the claimed identity.

(c) Using (a) and (b) one obtains $U_j^{\mathrm{row}} - u_j + 1 = \frac{1}{k_j^*}\left(D - k_j^* + 1 - B\right) + 1 = \frac{D-B+1}{k_j^*}$, which is a natural number as the number of subsignals was required to be divisible by k_L^*, implying divisibility by k_j^*.

(d) This is proved by induction for j. For $j = 0$, the left-hand side equals $\mathrm{Subsignal}_B(\xi,\ i)_\mu = \xi_{i+\mu-1}$ using Definition 1 for all $i \in \{1, \dots, D - B + 1\}$ and all $\mu \in \{1, \dots, B\}$. Since $k_0^* = 1$, Proposition 13 shows that the right-hand side equals $\xi_{\mathrm{div}(i-1,\ 1)+\mu,\ \mathrm{rem}(i-1,\ 1)+1} = \xi_{i-1+\mu,\ 1}$, hence both sides are equal.

Turning now to $j - 1 \to j$, let $\mu \in \{1, \dots, u_j\}$ be arbitrary, let $i \in \{1, \dots, D - B + 1\}$ be a fixed subsignal index and write $\tau := \mathrm{EvalStride}_{j-1}(\mathrm{Subsignal}_B(\xi,\ i)) \in M_{j-1}^{u_{j-1}}$ as an abbreviation. The left-hand side of the claim leads to

$$\begin{aligned}
&\mathrm{EvalStride}_j(\mathrm{Subsignal}_B(\xi,\ i))_\mu \\
\overset{\text{D. 22}}{=}\ &\mathrm{Stride}_{g_j}(T_j(\tau))_\mu \\
\overset{\text{D. 11}}{=}\ &g_j\left(\sum\nolimits_{\nu=1}^{k_j} T_j(\tau)_{k_j(\mu-1)+\nu} \cdot e_\nu^{k_j}\right) \\
\overset{\text{T. 6}}{=}\ &g_j\left(\sum\nolimits_{\nu=1}^{k_j} T_j\left(\sum\nolimits_{\lambda=1}^{c_j} \tau_{k_j(\mu-1)+\nu+\lambda-1} \cdot e_\lambda^{c_j}\right) \cdot e_\nu^{k_j}\right) \\
\overset{\text{IH}}{=}\ &g_j\left(\sum\nolimits_{\nu=1}^{k_j} T_j\left(\sum\nolimits_{\lambda=1}^{c_j} \mathrm{EvalSlide}_{j-1}(\xi)_{\psi,\ \omega} \cdot e_\lambda^{c_j}\right) \cdot e_\nu^{k_j}\right),
\end{aligned}$$

where $\psi := \mathrm{div}(i-1,\ k_{j-1}^*) + k_j(\mu-1) + \nu + \lambda - 1 \in \mathbb{N}_1$ and $\omega := \mathrm{rem}(i-1,\ k_{j-1}^*) + 1 \in \mathbb{N}_1$ are abbreviations.

Let $\pi := \mathrm{EvalSlide}_{j-1}(\xi) \in M_{j-1}^{U_{j-1}^{\mathrm{row}} \times U_{j-1}^{\mathrm{col}}}$ be an abbreviation for the analysis of the right-hand side of the claim. Now

$$\begin{aligned}
&\mathrm{EvalSlide}_j(\xi)_{\mathrm{div}(i-1,\ k_j^*)+\mu,\ \mathrm{rem}(i-1,\ k_j^*)+1} \\
\overset{\text{D. 22}}{=}\ &\mathrm{Frag}_{k_j}(\mathrm{Slide}_{g_j}(T_j(\pi))_{\mathrm{div}(i-1,\ k_j^*)+\mu,\ \mathrm{rem}(i-1,\ k_j^*)+1} \\
\overset{\text{L. 17}}{=}\ &\mathrm{Slide}_{g_j}(T_j(\pi))_{\mathrm{div}(\phi,\ k_{j-1}^*)+1,\ \mathrm{rem}(\phi,\ k_{j-1}^*)+1},
\end{aligned}$$

where the number of input fragments to Frag_{k_j} was k_{j-1}^*, as already shown in (b), and where it has been defined that $\phi := \left(\mathrm{div}(i-1,\ k_j^*) + \mu - 1\right) k_j k_{j-1}^* + \mathrm{rem}(i-1,\ k_j^*) \in \mathbb{N}$. By the definition of the operators from Euclidean division follows that $\phi = i - 1 + (\mu-1)k_j k_{j-1}^*$. Proposition 13 implies $\mathrm{div}(\phi,\ k_{j-1}^*) = \mathrm{div}(i-1,\ k_{j-1}^*) + k_j(\mu-1)$ and $\mathrm{rem}(\phi,\ k_{j-1}^*) + 1 = \mathrm{rem}(i-1,\ k_{j-1}^*) + 1 = \omega$. Hence

$$\begin{aligned}
&\mathrm{EvalSlide}_j(\xi)_{\mathrm{div}(i-1,\ k_j^*)+\mu,\ \mathrm{rem}(i-1,\ k_j^*)+1} \\
\overset{\text{D. 5}}{=}\ &g_j\left(\sum\nolimits_{\nu=1}^{k_j} T_j(\pi)_{\mathrm{div}(i-1,\ k_{j-1}^*)+k_j(\mu-1)+\nu,\ \omega} \cdot e_\nu^{k_j}\right) \\
\overset{\text{T. 6}}{=}\ &g_j\left(\sum\nolimits_{\nu=1}^{k_j} T_j\left(\sum\nolimits_{\lambda=1}^{c_j} \pi_{\psi,\ \omega} \cdot e_\lambda^{c_j}\right) \cdot e_\nu^{k_j}\right),
\end{aligned}$$

which equals the left-hand side of the claim as shown earlier and thus the proof is finished. □

In conclusion, the result of the EvalStride operator applied to an arbitrary subsignal of an input signal emerges contiguously in the result of the EvalSlide operator which processes the signal in its entirety. The concrete position in the fragmentation data structure can be determined with Lemma 23(d). An example for this is depicted in Fig. 5. It will be shown later how defragmentation can be used eventually to restore the expected order of the resulting samples. It is further intuitively clear that EvalSlide is much more efficient than EvalStride since redundant computations are avoided. This is analyzed rigorously in Sect. V.

Before continuing with the theory, a short discussion of two notable special cases. First, a processing chain is of course not required to have a pooling layer following every subsignal compatible transformation, or to even have pooling layers at all. Consider a fixed layer index $j \in \{1, \dots, L\}$. By setting $k_j := 1$ and $g_j\colon N_j \to M_j,\ \tau \mapsto \tau$, where $N_j = M_j$, one sees that Stride_{g_j}, Slide_{g_j} and Frag_{k_j} are the identity functions on their respective domains. In other words, this parameterization of pooling layer j causes it to act like a neutral bypass operation. If, on the other hand, one would want to have two pooling layers one directly after the other, it is completely analogous to achieve a neutral subsignal compatible transformation.

The other special case is to have a convolution which is evaluated in a non-overlapping manner, or equivalently in a strided fashion. This can be achieved by plugging f_{conv} from Sect. III-C into a pooling kernel within a processing chain. Non-overlapping convolution has, however, no advantage in computational complexity if entire input signals should be processed using a sliding window approach without accuracy loss: Lemma 23 states that strided fashion has then to be turned into sliding fashion, which is essentially the same as carrying out convolution conventionally by advancing the filter banks exactly one sample after each evaluation.

C. Defragmentation and Arbitrary Input Signal Length

Lemma 23 requires the length of the input signal to satisfy certain divisibility constraints. Extension of its statements to signals of arbitrary length requires two additional operators:

Definition 24. Let $r \in \mathbb{N}$ be a natural number, let M be a set and $\zeta \in M$ be an arbitrary dummy element from M. Then $\mathrm{Stuff}_r\colon \cup_1(M) \to \cup_{r+1}(M)$,

$$(\xi_1, \dots, \xi_q) \mapsto \sum\nolimits_{\nu=1}^q \xi_\nu \cdot e_\nu^{q+r} + \sum\nolimits_{\nu=1}^r \zeta \cdot e_{q+\nu}^{q+r},$$

is called the *stuffing operator*, which appends r copies of ζ to its argument. Further, $\mathrm{Trim}_r\colon \cup_{r+1}(M) \to \cup_1(M)$,

$$(\xi_1, \dots, \xi_q, \xi_{q+1}, \dots, \xi_{q+r}) \mapsto \sum\nolimits_{\nu=1}^q \xi_\nu \cdot e_\nu^q,$$

is called the *trimming operator*, which removes the final r entries from its argument.

The concrete choice of the dummy element ζ does not matter in the following considerations since all output entries which are affected by its choice are trimmed away in the end. It is now possible to state the main theoretical result of this section:

Theorem 25. Consider a processing chain with the same notation as in Definition 22, where the application in a strided fashion is well-defined as in Lemma 23. Further assume that $\dim_{M_L}(\mathrm{EvalStride}_L(\rho)) = 1$ for all $\rho \in M_0^B$, that is the output of the entire processing chain applied in a strided fashion consists of exactly one sample.

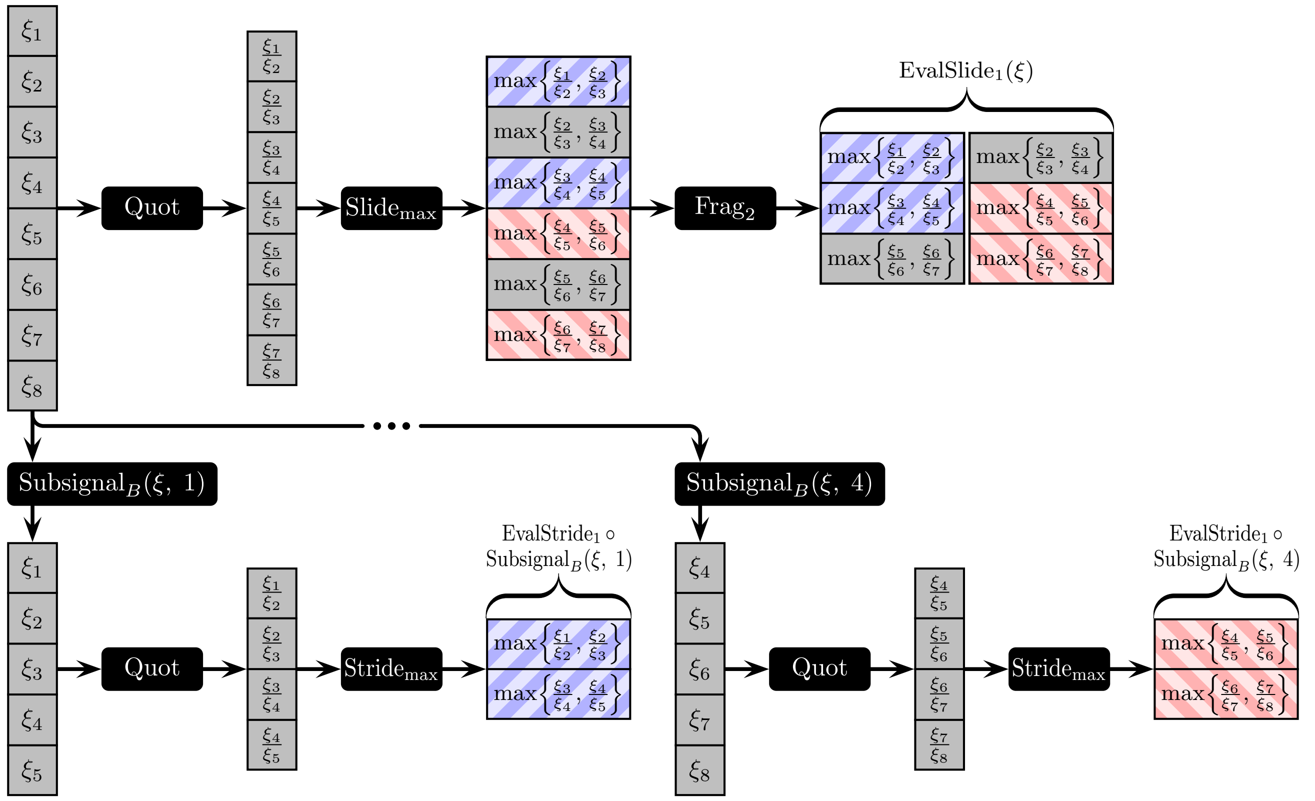


Fig. 5. Example of the two notions of a one-layered ($L = 1$) processing chain with receptive field size $B = 5$ applied to an input signal ξ with $D = 8$ samples. The processing chain consists of the Quot operator as subsignal compatible transformation followed by max-pooling with a stride of $k_1 = 2$. In this example, the processing chain is well-defined and all divisibility requirements are met. The upper part shows the result of EvalSlide evaluated on ξ: Here, max-pooling is applied to Quot(ξ) in a sliding fashion, followed by fragmentation producing two fragments. The lower part shows the outcome of EvalStride applied to two different subsignals of length B. Max-pooling is used here in a strided fashion which halves dimensionality. It is evident that the outcome of EvalStride can be found within Slide$_{\max}$ (Quot(ξ)), although the individual output signals are interleaved (highlighted with a colored pattern in the graphics). The reordering through fragmentation corrects interleaving so that the outcome is always available contiguously. A second layer in the processing chain has then the ability to further process independent contiguous signals, which is particularly effective on real computing machines.

Let $\tilde{r}\colon \mathbb{N}_1 \to \{0, \dots, k_L^* - 1\}$,

$$\delta \mapsto \begin{cases} 0, & \text{if } k_L^* \text{ divides } \delta - B + 1, \\ k_L^* - \operatorname{rem}(\delta - B + 1,\ k_L^*), & \text{otherwise,} \end{cases}$$

denote the number of dummy samples that have to be padded to an original signal with δ samples to satisfy divisibility requirements. Further define $r\colon \cup_1 (M_0) \to \{0, \dots, k_L^* - 1\}$, $\xi \mapsto \tilde{r}(\dim_{M_0}(\xi))$, as an abbreviation that computes the required number of dummy samples in dependence on a concrete original signal ξ. Consider $T\colon \cup_B (M_0) \to \cup_1(M_L)$,

$$\xi \mapsto \operatorname{Trim}_{r(\xi)}(\operatorname{Defrag}_{k_L^*}(\operatorname{EvalSlide}_L(\operatorname{Stuff}_{r(\xi)}(\xi)))).$$

This function first stuffs the input signal with as many dummy samples such that each fragmentation operation during application of the processing chain in a sliding fashion comes out even, applies the processing chain in a sliding fashion, defragments the outcome and eventually removes all superfluous entries that emerged from the initial stuffing. Then T is a subsignal compatible transformation with dimensionality reduction constant B. Furthermore, $T(\rho) = \operatorname{EvalStride}_L(\rho)$ for all $\rho \in M_0^B$ and $T = \operatorname{Slide}_{\operatorname{EvalStride}_L}$.

Proof. Note that $\tilde{r}(\delta)$ is well-defined if k_L^* does not divide $\delta - B + 1$, since then $\operatorname{rem}(\delta - B + 1,\ k_L^*) \in \{1, \dots, k_L^* - 1\}$ and thus $k_L^* - \operatorname{rem}(\delta - B + 1,\ k_L^*) \in \{1, \dots, k_L^* - 1\}$.

Let $\xi \in \cup_B(M_0)$ and write $\tilde{D} := \dim_{M_0}(\xi)$. Then $D := \dim_{M_0}(\operatorname{Stuff}_{r(\xi)}(\xi)) = \tilde{D} + r(\xi)$. If k_L^* divides $\tilde{D} - B + 1$ it is $r(\xi) = 0$, so $\operatorname{rem}(\dim_{M_0}(\operatorname{Stuff}_{r(\xi)}(\xi)) - B + 1,\ k_L^*) = 0$. If on the other hand k_L^* does not divide $\tilde{D} - B + 1$, then $r(\xi) > 0$ and $\dim_{M_0}(\operatorname{Stuff}_{r(\xi)}(\xi)) - B + 1 = \tilde{D} + k_L^* - \operatorname{rem}(\tilde{D} - B + 1,\ k_L^*) - B + 1$. Hence $\operatorname{rem}(\dim_{M_0}(\operatorname{Stuff}_{r(\xi)}(\xi)) - B + 1,\ k_L^*) = 0$ due to the idempotence of the rem operator. Therefore, the number of subsignals of $\operatorname{Stuff}_{r(\xi)}(\xi)$ with B samples is always divisible by k_L^*, as required for application of Lemma 23.

Lemma 23 guarantees that $\pi := \operatorname{EvalSlide}_L(\operatorname{Stuff}_{r(\xi)}(\xi))$ is well-defined. With Lemma 23(b) follows $\operatorname{cdim}_{M_L}(\pi) = k_L^*$. Since $\dim_{M_L}(\operatorname{EvalStride}_L(\rho)) = 1$ was required for all $\rho \in M_0^B$, $\operatorname{rdim}_{M_L}(\pi) = \frac{1}{k_L^*}(D - B + 1)$ with Lemma 23(c). Therefore, $\operatorname{Defrag}_{k_L^*}(\pi)$ is well-defined with exactly one output fragment, which has the same number of samples as there are subsignals of length B in the stuffed input signal: $\dim_{M_L}(\operatorname{Defrag}_{k_L^*}(\pi)) = D - B + 1$.

Now $\dim_{M_L}(\operatorname{Defrag}_{k_L^*}(\pi)) \geq r(\xi) + 1$ holds because $D = \tilde{D} + r(\xi)$ where $\tilde{D} \geq B$, thus $\operatorname{Trim}_{r(\xi)}(\operatorname{Defrag}_{k_L^*}(\pi))$ is well-defined. Since trimming reduces dimensionality by $r(\xi)$ follows $\dim_{M_0}(\xi) - \dim_{M_L}(T(\xi)) = \tilde{D} - (D - B + 1 - r(\xi)) = B - 1$. Therefore, T fulfills the dimensionality reduction property with dimensionality reduction constant B.

To prove T is subsignal compatible it is hence sufficient to use Theorem 6 to show $T(\mathrm{Subsignal}_B(\xi,\ i)) = T(\xi)_i \in M_L$ for all $i \in \{1, \dots, \tilde{D} - B + 1\}$. It is first demonstrated that $T(\rho) = \mathrm{EvalStride}_L(\rho)$ for all $\rho \in M_0^B$. This can then be used to prove the weakened exchange property.

Let $\rho \in M_0^B$, then $\mathrm{Subsignal}_B(\mathrm{Stuff}_{r(\rho)}(\rho),\ 1) = \rho$ by the definition of the stuffing operator. $T(\rho)$ consists of a single sample since the dimensionality reduction constant of T is B, hence $T(\rho) = T(\rho)_1$. Extraction of the very first sample of the result of the trimming operator is equal here to the extraction of the very first sample of the trimming operator's argument. Therefore,

$$\begin{aligned} & T(\rho) \\ = \quad & \mathrm{Defrag}_{k_L^*}(\mathrm{EvalSlide}_L(\mathrm{Stuff}_{r(\rho)}(\rho)))_{1,\ 1} \\ \overset{\text{L. 18}}{=} \quad & \mathrm{EvalSlide}_L(\mathrm{Stuff}_{r(\rho)}(\rho))_{\mathrm{div}(0,\ k_L^*)+1,\ \mathrm{rem}(0,\ k_L^*)+1} \\ \overset{\text{L. 23(d)}}{=} \quad & \mathrm{EvalStride}_L(\mathrm{Subsignal}_B(\mathrm{Stuff}_{r(\rho)}(\rho),\ 1))_1 \\ = \quad & \mathrm{EvalStride}_L(\rho). \end{aligned}$$

Returning to the exchange property, let $\xi \in M_0^{\tilde{D}}$ as before, and let $i \in \{1, \dots, \tilde{D} - B + 1\}$ be an arbitrary subsignal index. Omitting the trimming operator as before yields

$$\begin{aligned} & T(\xi)_i \\ = \quad & \mathrm{Defrag}_{k_L^*}(\mathrm{EvalSlide}_L(\mathrm{Stuff}_{r(\xi)}(\xi)))_{i,\ 1} \\ \overset{\text{L. 18}}{=} \quad & \mathrm{EvalSlide}_L(\mathrm{Stuff}_{r(\xi)}(\xi))_{\mathrm{div}(i-1,\ k_L^*)+1,\ \mathrm{rem}(i-1,\ k_L^*)+1} \\ \overset{\text{L. 23(d)}}{=} \quad & \mathrm{EvalStride}_L(\mathrm{Subsignal}_B(\mathrm{Stuff}_{r(\xi)}(\xi),\ i))_1 \\ \overset{\text{D. 24}}{=} \quad & \mathrm{EvalStride}_L(\mathrm{Subsignal}_B(\xi,\ i)) \\ = \quad & T(\mathrm{Subsignal}_B(\xi,\ i)). \end{aligned}$$

Hence T is a subsignal compatible transformation due to Theorem 6. Theorem 4 finally implies $T = \mathrm{Slide}_{\mathrm{EvalStride}_L}$. $\square$

It can be concluded that CNNs can be turned into efficiently computable subsignal compatible transformations using the $\mathrm{EvalSlide}$ operator regardless of the input signal's dimensionality. One could suspect that stuffing the input signal with dummy samples might have a negative effect on the efficiency. However, the number of stuffed samples is always less than the stride product k_L^* of the final layer and hence very small for reasonably sized CNNs.

Moreover, stuffing guarantees that all fragments encountered during evaluation are homogeneous. This enables tensors to be used as the sole data structure for input data, intermediate representations and computation results. This is far more efficient than storing each fragment individually, especially on massively parallel processors where then simple parallelized implementations can achieve maximum throughput.

V. Computational Complexity Analysis

In this section, a detailed theoretical analysis of the computational complexity of processing chain evaluation is carried out. As introduced in Definition 22, this corresponds to the alternating application of subsignal compatible transformations and functions applied in a strided fashion. For measuring the computational complexity of the $\mathrm{EvalStride}$ and $\mathrm{EvalSlide}$ operators, the function evaluations required for computing the output of each layer are counted. Regarding the subsignal compatible transformations, the unique functions that generate the transformations on account of Theorem 6 are considered.

It is shown that $\mathrm{EvalSlide}$ requires at most the same number of function evaluations as $\mathrm{EvalStride}$ in each layer. This then implies that $\mathrm{EvalSlide}$ is more efficient on the global scale of an entire processing chain where all the layers are evaluated subsequently. Further, it is shown that the theoretical speedup can be factorized into simple expressions, facilitating statements on the effect of individual parameters.

A. *Identification of the Number of Function Evaluations*

Assume a situation of Lemma 23 in which an input signal $\xi \in M_0^D$ and a well-defined processing chain are given. An arbitrary layer index $j \in \{1, \dots, L\}$ is fixed and $\mathrm{EvalStride}_j$ is analyzed for an arbitrary subsignal index $i \in \{1, \dots, D - B + 1\}$. As in the proof of Lemma 23(d), write $\tau := \mathrm{EvalStride}_{j-1}(\mathrm{Subsignal}_B(\xi,\ i)) \in M_{j-1}^{u_{j-1}}$ which results in $\mathrm{EvalStride}_j(\mathrm{Subsignal}_B(\xi,\ i)) = \mathrm{Stride}_{g_j}(T_j(\tau))$. Now Theorem 6 guarantees that there is exactly one function $f_j \colon M_{j-1}^{c_j} \to N_j$ satisfying $T_j = \mathrm{Slide}_{f_j}$. Thus

$$T_j(\tau) \overset{\text{D. 5}}{=} \sum\nolimits_{\mu=1}^{u_{j-1}-c_j+1} f_j(\mathrm{Subsignal}_{c_j}(\tau,\ \mu)) \cdot e_\mu^{u_{j-1}-c_j+1},$$

hence f_j has to be evaluated $u_{j-1} - c_j + 1$ times. Considering the strided application of g_j, one notes $u_{j-1} - c_j + 1 = k_j u_j$ with Lemma 23(a) and furthermore

$$\begin{aligned} & \mathrm{Stride}_{g_j}(T_j(\tau)) \\ \overset{\text{D. 11}}{=} \; & \sum\nolimits_{\nu=1}^{u_j} g_j(\mathrm{Subsignal}_{k_j}(T_j(\tau),\ k_j(\nu-1)+1)) \cdot e_\nu^{u_j}. \end{aligned}$$

Here, u_j evaluations of g_j are necessary. Since all function evaluations have to be carried out for each of the $D - B + 1$ possible subsignals, the total number of function evaluations increases proportionately with this factor.

Redundant computations are avoided if $\mathrm{EvalSlide}_j$ is used instead. Here the complexity of processing an individual fragment in layer j is analyzed. The overall complexity then results from multiplication with the number of input fragments U_{j-1}^{col}. Analogous to the proof of Lemma 23(d), let $\gamma \in \{1, \dots, U_{j-1}^{\mathrm{col}}\}$ be a fragment index and define

$$\pi := \sum\nolimits_{\kappa=1}^{U_{j-1}^{\mathrm{row}}} \mathrm{EvalSlide}_{j-1}(\xi)_{\kappa,\ \gamma} \cdot e_\kappa^{U_{j-1}^{\mathrm{row}}} \in M_{j-1}^{U_{j-1}^{\mathrm{row}}}$$

as an abbreviation for the input fragment with index γ. Now, the output of the j-th layer of the processing chain is

$$\mathrm{EvalSlide}_j(\xi) = \mathrm{Frag}_{k_j}(\mathrm{Slide}_{g_j}(T_j(\mathrm{EvalSlide}_{j-1}(\xi)))).$$

The complexity of the fragmentation operator is neglected here because it is merely a structured permutation with very little overhead. Considering a single fragment now yields

$$T_j(\pi) \overset{\text{D. 5}}{=} \sum\nolimits_{\mu=1}^{U_{j-1}^{\mathrm{row}}-c_j+1} f_j(\mathrm{Subsignal}_{c_j}(\pi,\ \mu)) \cdot e_\mu^{U_{j-1}^{\mathrm{row}}-c_j+1},$$

accounting for $U_{j-1}^{\mathrm{row}} - c_j + 1$ evaluations of f_j. Application of g_j in a sliding fashion yields

$$\mathrm{Slide}_{g_j}(T_j(\pi)) \overset{\text{D. 5}}{=} \sum\nolimits_{\nu=1}^{\chi} g_j(\mathrm{Subsignal}_{k_j}(T_j(\pi),\ \nu)) \cdot e_\nu^\chi,$$

which requires $\chi := U_{j-1}^{\mathrm{row}} - c_j - k_j + 2$ evaluations of g_j. Lemma 23(b) finally implies $\chi = k_j U_j^{\mathrm{row}}$.

B. Analysis for the Subsignal Compatible Transformation Evaluation Component

To determine the resulting speedup when redundant computations are avoided, the ratio of the number of function evaluations required for the naive approach $\mathrm{EvalStride}_j$ to the number needed when the input signal is processed in its entirety using $\mathrm{EvalSlide}_j$ is evaluated. Considering f_j this yields

$$S_{f_j} := \frac{(D-B+1)(u_{j-1}-c_j+1)}{U_{j-1}^{\mathrm{col}}(U_{j-1}^{\mathrm{row}}-c_j+1)},$$

where the number of subsignals and the number of fragments was included in the numerator and denominator, respectively. With Lemma 23(b) and Lemma 23(c) one obtains $\frac{D-B+1}{U_{j-1}^{\mathrm{col}}} = U_{j-1}^{\mathrm{row}} - u_{j-1} + 1$, and by substituting this and after minor algebraic manipulation one sees that

$$S_{f_j} = 1 + \frac{(U_{j-1}^{\mathrm{row}} - u_{j-1})(u_{j-1}-c_j)}{U_{j-1}^{\mathrm{row}} - c_j + 1}.$$

Since $1 \le c_j \le u_{j-1} \le U_{j-1}^{\mathrm{row}}$ using Lemma 23(c) it follows that $S_{f_j} \ge 1$, which means $\mathrm{EvalSlide}_j$ requires at most the same number of applications of f_j as $\mathrm{EvalStride}_j$. Merely in the special cases where the extent u_{j-1} of the region fed into layer j equals the length of the signal fragments ($u_{j-1} = U_{j-1}^{\mathrm{row}}$) or the dimensionality reduction constant of the subsignal compatible transformation ($u_{j-1} = c_j$) the speedup attains unity, indicating that both approaches require the same number of function evaluations.

If S_{f_j} is understood as a function dependent upon the signal dimensionality D, then the only quantity that depends on D in the derived expression is U_{j-1}^{row} since c_j is constant and u_{j-1} is independent of D as can be seen from Lemma 23(a). As Lemma 23 requires $D - B + 1$ to be divisible by k_L^*, the next larger feasible signal dimensionality is $D + k_L^* =: D_+$. Subtracting S_{f_j} evaluated for signal dimensionality D from its value for an extended signal dimensionality D_+ yields

$$\begin{aligned} &S_{f_j}(D_+) - S_{f_j}(D) \\ &= \frac{(u_{j-1}-c_j)(u_{j-1}-c_j+1)(U_{j-1}^{\mathrm{row}}(D_+) - U_{j-1}^{\mathrm{row}}(D))}{(U_{j-1}^{\mathrm{row}}(D_+) - c_j + 1)(U_{j-1}^{\mathrm{row}}(D) - c_j + 1)}. \end{aligned}$$

Now Lemma 23(b) implies that $U_{j-1}^{\mathrm{row}}(D_+) - U_{j-1}^{\mathrm{row}}(D) = \frac{k_L^*}{k_{j-1}^*}$, therefore $S_{f_j}(D_+) \ge S_{f_j}(D)$ and thus the speedup increases if the signal dimensionality is increased. In the limit case of arbitrarily large input signals one obtains $\lim_{D\to\infty} S_{f_j}(D) = u_{j-1} - c_j + 1$, that is the speedup asymptotically attains a finite value. From this it is evident that greatest speedups can be achieved for large regions of interest u_{j-1} and small dimensionality reduction constants c_j.

C. Analysis for the Strided Function Evaluation Component

Turning now to the function g_j applied in a strided fashion, incorporating the number of subsignals and fragments for $\mathrm{EvalStride}_j$ and $\mathrm{EvalSlide}_j$, respectively, yields the ratio

$$S_{g_j} := \frac{(D-B+1)u_j}{U_{j-1}^{\mathrm{col}} k_j U_j^{\mathrm{row}}} = 1 + \frac{(U_j^{\mathrm{row}} - u_j)(u_j - 1)}{U_j^{\mathrm{row}}}.$$

With Lemma 23(c) follows $S_{g_j} \ge 1$, hence here the complexity of $\mathrm{EvalSlide}_j$ is also less than that of $\mathrm{EvalStride}_j$. The speedup is unity only if $u_j = U_j^{\mathrm{row}}$ or $u_j = 1$.

Grasping S_{g_j} as a function of D and denoting the next larger signal dimensionality with $D_+ := D + k_L^*$, one obtains

$$S_{g_j}(D_+) - S_{g_j}(D) = \frac{u_j(u_j-1)(U_j^{\mathrm{row}}(D_+) - U_j^{\mathrm{row}}(D))}{U_j^{\mathrm{row}}(D_+)U_j^{\mathrm{row}}(D)} \ge 0.$$

Thus speedups increase with larger signal dimensionality. In the limit case of arbitrarily large input signals it is $\lim_{D\to\infty} S_{g_j}(D) = u_j$, hence here the speedup is bounded from above as well.

D. Discussion

Simple expressions have been derived that imply the number of function evaluations required by EvalSlide is always less or equal than that for EvalStride in each layer. Therefore this also holds for the subsequent evaluation of all the layers. Further, the special cases have been identified in which the computational complexity of EvalSlide matches that of EvalStride. Moreover, it was demonstrated that the speedup becomes more significant for larger input signals, although its growth is not unbounded.

The analysis was restricted to only the amount of necessary function evaluations and neglected the parallelization potential of the individual approaches. On a massively parallel processor, the throughput of the EvalSlide approach might be substantially lower than that of EvalStride since here coarse-grained parallelism on the subsignal level can be exploited facilitating load balancing on thousands of parallel computing cores. However, experiments in the next section demonstrate that, as predicted by the theoretical analysis, in practice EvalSlide is orders of magnitude faster than EvalStride.

VI. Practical Considerations for Image Processing and Experimental Evaluation

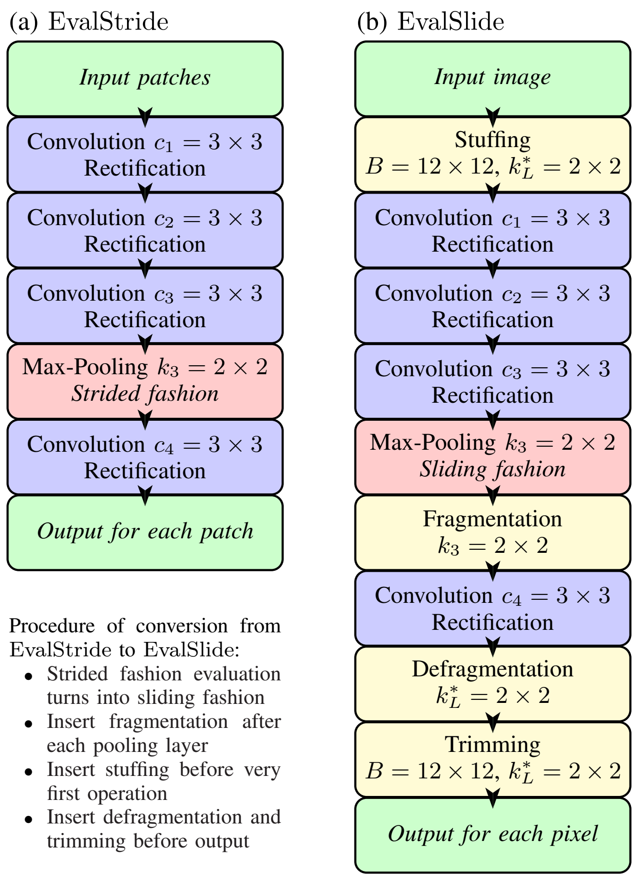
This section discusses practical considerations when using the theory proposed in this paper for image processing tasks. Further, results of experiments on semantic segmentation and runtime measurements on real processors are reported.

A. Generalization to 2D Image Data and Transformation for Image-Based Application

In the context of image processing, two-dimensional signals are referred to as *images* and two-dimensional subsignals as *patches*. The generalization of subsignal compatible transformation theory to two spatial dimensions is straightforward and shown here exemplarily for Definition 3. Images are represented by matrices, patch indices are two-dimensional and patch extraction is the same as forming a submatrix of adjacent entries. Now suppose T is a transformation from the space of sufficiently large images ξ with pixels in the set M to images with pixels from the set N. Then T fulfills the dimensionality reduction property with dimensionality reduction constants $r, c \in \mathbb{N}_1$ if both $\mathrm{rdim}_N(T(\xi)) = \mathrm{rdim}_M(\xi) - r + 1$ and $\mathrm{cdim}_N(T(\xi)) = \mathrm{cdim}_M(\xi) - c + 1$. The exchange property generalizes to $T(\text{Patch}_{d_r \times d_c}(\xi,\ i,\ j)) = \text{Patch}_{(d_r - r + 1) \times (d_c - c + 1)}(T(\xi),\ i,\ j)$, which is a condition in two dimensions. Here, Patch is the patch extraction operator with subscripts specifying the dimensionalities of the extracted patches, $d_r, d_c \in \mathbb{N}_1$, $d_r \geq r$, $d_c \geq c$, are patch dimensionalities and $i \in \{1, \dots, \text{rdim}_M(\xi) - d_r + 1\}$ and $j \in \{1, \dots, \text{cdim}_M(\xi) - d_c + 1\}$ are patch indices. If both properties are fulfilled, T is called a *patch compatible transformation*.

The remaining theory can be generalized analogously. In practice it is sufficient to use plain 4D tensors as the only data structure for CNN input data, intermediate representations and computation results. Here, one dimension accounts for the feature map index and two dimensions account for the spatial position within each feature map. The fourth dimension represents an image index. Although image processing fragmentation requires two dimensions, these can be collapsed into one dimension by linearizing the two-dimensional fragment indices. Since all computations on fragments are carried out independently by Definition 21, this essentially corresponds to the meaning of the image index dimension in common 4D tensor processing [22]. Therefore, fragmentation does not require any modifications whatsoever to the computationally demanding routines such as tensor convolution.

For an experimental evaluation, CNNs with a varying number of convolutional layers L were created using the following scheme. Similar to [23], each convolutional layer was parameterized for a filter size of 3×3 pixels. The output of each convolutional layer was fed through a rectification non-linearity [24]. A 2×2 max-pooling layer was inserted after each three pairs of convolution and rectification layers, unless the pooling layer would be the final layer of the network [23]. Fig. 6 depicts such a network architecture as it was used for the EvalStride operator and how it was transformed to account for image-based application using the EvalSlide operator (see Definition 22 for the formal recipe).



(a) EvalStride
Input patches
Convolution $c_1 = 3 \times 3$ Rectification
Convolution $c_2 = 3 \times 3$ Rectification
Convolution $c_3 = 3 \times 3$ Rectification
Max-Pooling $k_3 = 2 \times 2$ *Strided fashion*
Convolution $c_4 = 3 \times 3$ Rectification
Output for each patch
(b) EvalSlide
Input image
Stuffing $B = 12 \times 12$, $k_L^* = 2 \times 2$
Convolution $c_1 = 3 \times 3$ Rectification
Convolution $c_2 = 3 \times 3$ Rectification
Convolution $c_3 = 3 \times 3$ Rectification
Max-Pooling $k_3 = 2 \times 2$ *Sliding fashion*
Fragmentation $k_3 = 2 \times 2$
Convolution $c_4 = 3 \times 3$ Rectification
Defragmentation $k_L^* = 2 \times 2$
Trimming $B = 12 \times 12$, $k_L^* = 2 \times 2$
Output for each pixel

Procedure of conversion from EvalStride to EvalSlide:
- Strided fashion evaluation turns into sliding fashion
- Insert fragmentation after each pooling layer
- Insert stuffing before very first operation
- Insert defragmentation and trimming before output

Fig. 6. Example network architecture for $L = 4$ targeted at the processing of two-dimensional images. For EvalStride (left-hand side) suitable for patch-based application, this architecture consists of four 3×3 convolutional layers, where after the third convolutional layer 2×2 max-pooling is carried out in a strided fashion. The right-hand side shows how this architecture has been transformed for the EvalSlide approach suitable for image-based application: Pooling is now applied in a sliding fashion and followed by fragmentation. The input image is stuffed to guarantee fragmentation always comes out even and hence homogeneous 4D tensors can be used as the only data structure. The effects of stuffing and fragmentation are undone in the end, yielding an output for each pixel in the input image. Here, B denotes the entire CNN's receptive field size and k_L^* is the stride product of the entire network.

B. Semantic Image Segmentation

The practicality of the approach proposed in this paper has been verified by realizing a semantic image segmentation through evaluation of a classifier on all feasible patches in an image [11], [12], [13]. In doing so, images recorded with a wide angle camera attached to the windshield of an experimental vehicle were manually labeled to yield regions which only contain pixels of the four object categories road, vehicle, person, and background. As a classifier, a CNN with $L = 12$ convolutional layers as described above was used. The number of output feature maps was set to 16 for the first three convolutional layers and subsequently doubled after each pooling layer. A fully-connected layer (spatial dimension 1×1) with four output feature maps was appended to project from the high-dimensional feature space into the label space, followed by a final softmax non-linearity [20].

The CNN was first trained with patches extracted from random positions of the images from the learning set (EvalStride notion), transformed to image-based application, and thereafter fine-tuned using entire images (EvalSlide notion). Therefore, a huge number of weight updates on unbiased learning examples was carried out in the early phase of training, facilitating fast learning progress [25]. After the transformation, learning on entire images ensured all feasible patches were considered, improving homogeneity and reducing remaining misclassification artifacts. Note that backpropagating gradients through a transformed CNN is straightforward: The gradients of stuffing and trimming are trivial, and fragmentation and defragmentation are merely inversely related permutations. Elementary calculus is sufficient for determination of the gradient of max-pooling in a sliding fashion.

The classification decisions on an image from the test set just before defragmentation is carried out are shown in Fig. 7. Since for $L = 12$ three 2×2 pooling layers are involved, accounting for a stride product of $k_L^* = 8 \times 8$, there are 64 fragments in total. Each of the fragments represents a low-resolution version of the final output shifted by the corresponding number of pixels in either spatial dimension. Defragmentation yields a single output image, see Fig. 8,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSP.2016.2631454, IEEE Transactions on Signal Processing

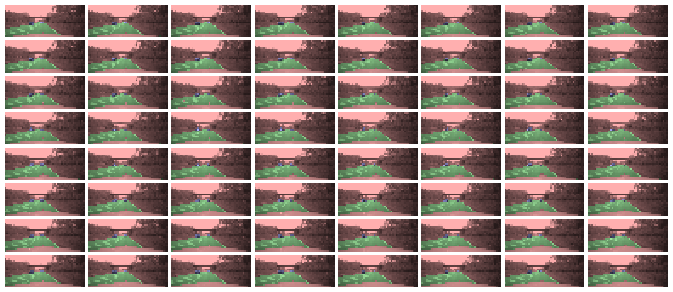


Fig. 7. Visualization of the fragmented classification decisions. Each fragment carries only low-resolution information, which is afterwards combined with a defragmentation operation (see Fig. 8 for the result).

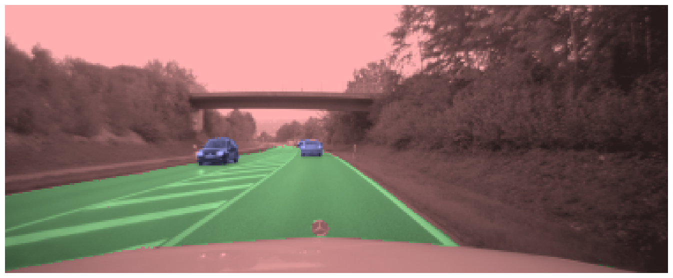


Fig. 8. Visualization of the final output of the classifier (best viewed in color). The grayscale input image has been tinted with the defragmented classification decisions. The information obtained facilitates an accurate representation of the scene in front of the vehicle.

where the classification decisions are available at high resolution. This output can subsequently be employed for vehicle environment perception [16]. It is however more efficient to additionally employ multi-scale analysis which incorporates context information and hence provides more discriminant features [13]. This technique was not considered in this paper due to space constraints. An elaborate discussion of its theoretical background is available in the technical report [26].

C. Runtime Measurements and Speedup Verification

Section V analyzed the theoretical speedup that can be expected if redundant computations are eliminated by means of the EvalSlide operator. To confirm whether speed can be significantly increased when real processors are employed, CNNs were applied to entire two-dimensional images using patch-based and image-based application. The CNNs were parameterized as described above, where the number of layers $L \in \{1, \dots, 15\}$ was varied and where the number of output feature maps was always set to 128. This renders the computational complexity of each layer about equal, allowing an undistorted evaluation of the overall speedup. All degrees of freedom of the CNNs were initialized with random numbers and random image data was used as input. This is no restriction to the generality or practicality of the results since the focus was on an analysis of runtime measurements and an assessment of the achieved speedups.

The experiments were run on a massively parallel GPU implementation and on a parallel CPU implementation. For the GPU variant, an NVIDIA GeForce GTX 980 Ti graphics card was used. Computations where sped up through

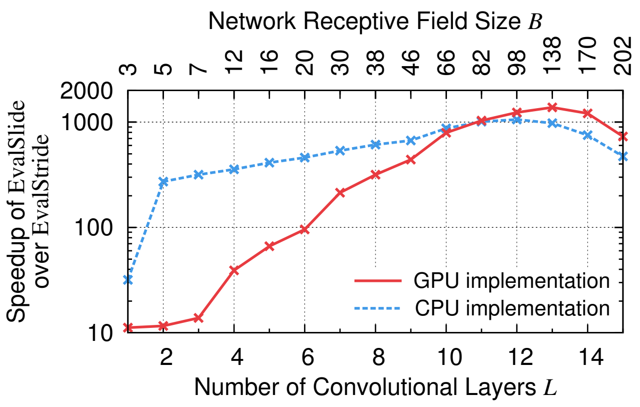


Fig. 9. Speedup factors of EvalSlide over EvalStride as measured on GPU and on CPU in dependence on the number of convolutional layers L in the networks. The second axis on top depicts the resulting receptive field size B of the entire network in either spatial dimension. Note that these values are non-linear in L since pooling layers were inserted at regular intervals in the networks. The measurements indicate that especially for deep networks, huge speedups can be obtained if images are processed in their entirety.

the cuDNN [22] software library. The CPU implementation employed an Intel Core i7-5930K processor. Here, the Intel Integrated Performance Primitives [27] and OpenMP [28] libraries were used for increasing efficiency.

The images had a height of 240 pixels, a width of 320 pixels and a single feature map was used as input for the networks. The time required for carrying out both operators on GPU and on CPU was measured and the ratio taken to determine the speedup. All measurements were repeated twenty times on twenty distinct input images, and the average of the resulting four hundred runs was used for further evaluation.

For EvalStride, neither the time required for extracting all feasible patches and storing them in a dedicated tensor nor the time required for assembling the final output tensor were included in the measurements. Due to memory constraints, the tensor with all the patches had to be broken down into batches before processing on the GPU. The batch size was maximized with respect to the available GPU memory to ensure maximum throughput of the graphics card. For EvalSlide, the time required for stuffing, fragmentation, defragmentation and trimming was included in the measurements. Here, splitting into batches was not necessary since redundancies in overlapping patches are avoided and the memory demands were therefore very low. Any measured speedups are hence biased in favor of EvalStride as here only computations but no overhead in organizing data structures were considered.

The achieved speedups both for GPU and CPU of EvalSlide over EvalStride in dependence on the number of convolutional layers L is depicted in Fig. 9. Although the input images were rather small, a notable speedup could be determined even for very shallow networks. Even for $L = 1$, significant speedups could be measured although here the theoretical number of function evaluations was equal for both operators. However, there was still a huge memory redundancy in the tensor storing all the patches necessary for EvalStride, which was disadvantageous in terms of memory throughput. While the CPU implementation achieved speedup factors beyond one hundred for $L \geq 2$, the GPU implementation required deeper networks with $L \geq 7$ for a similar speedup. This is because of the GPU's superior parallelization capabilities which facilitated an overproportional throughput of the $\mathrm{EvalStride}$ operator where the patches could be processed in parallel.

A peak in the speedup for $L = 13$ and $L = 12$ for GPU and CPU, respectively, was noted. If deeper networks were used, the relative speedup decreased but remained at a high level. The reason for this was the large receptive field size B of such deep networks: Since the patch size almost matched the image dimensions, there were only relatively few patches compared to the situation of a smaller receptive field size. As predicted by the theoretical results from Sect. V, the degree of redundancy of $\mathrm{EvalStride}$ decreases in this case resulting in a decreased relative speedup.

Finally, the execution times of $\mathrm{EvalSlide}$ using the GPU implementation versus the CPU implementation were compared. Averaging over $L \in \{1, \ldots, 15\}$ yielded a relative speedup of the GPU implementation of factor 89 over the CPU implementation, demonstrating the massive parallelization potential.

VII. Conclusions

This paper introduced and analyzed the concept of subsignal compatible transformations, functions that allow exchanging subsignal extraction with function evaluation without any effect on the outcome. In doing so, it was demonstrated how CNNs can be applied efficiently without accuracy loss to large signals using a sliding window approach and homogeneous data structures while eliminating redundant computations and special case treatment. A theoretical analysis has proven the computational complexity of processing an input signal in its entirety is inferior to subsignal-based application, which was subsequently verified through numerical experiments. All theoretical results have been proven rigorously, mathematically demonstrating the exactness of the proposed approach. The theoretical framework developed in this paper facilitates further research for gaining deeper insight into related methods for dense signal scanning.

References

[1] D. H. Hubel and T. N. Wiesel, "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex," *Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[2] K. Fukushima, "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," in *Advances in Neural Information Processing Systems*, vol. 2, 1990, pp. 396–404.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] V. Jain and H. S. Seung, "Natural Image Denoising with Convolutional Networks," in *Advances in Neural Information Processing Systems*, vol. 21, 2009, pp. 769–776.

[6] L. Xu, J. S. Ren, C. Liu, and J. Jia, "Deep Convolutional Neural Network for Image Deconvolution," in *Advances in Neural Information Processing Systems*, vol. 27, 2015, pp. 1790–1798.

[7] C. Dong, C. C. Loy, K. He, and X. Tang, "Image Super-Resolution Using Deep Convolutional Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[8] D. C. Cireşan, U. Meier, and J. Schmidhuber, "Multi-Column Deep Neural Networks for Image Classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2013, pp. 1097–1105.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[11] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano, "Toward Automatic Phenotyping of Developing Embryos from Videos," *IEEE Transactions on Image Processing*, vol. 14, no. 9, pp. 1360–1371, 2005.

[12] D. Grangier, L. Bottou, and R. Collobert, "Deep Convolutional Networks for Scene Parsing," in *International Conference on Machine Learning, Workshop on Learning Feature Hierarchies*, 2009.

[13] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning Hierarchical Features for Scene Labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.

[14] A. Giusti, D. C. Cireşan, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Fast Image Scanning with Deep Max-Pooling Convolutional Neural Networks," in *IEEE International Conference on Image Processing*, 2013, pp. 4034–4038.

[15] W. Thong, S. Kadoury, N. Piché, and C. J. Pal, "Convolutional Networks for Kidney Segmentation in Contrast-Enhanced CT Scans," *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 2016.

[16] D. Nuss, M. Thom, A. Danzer, and K. Dietmayer, "Fusion of Laser and Monocular Camera Data in Object Grid Maps for Vehicle Environment Perception," in *Proceedings of the International Conference on Information Fusion*, 2014.

[17] R. Vaillant, C. Monrocq, and Y. LeCun, "An Original Approach for the Localization of Objects in Images," in *Proceedings of the International Conference on Artificial Neural Networks*, 1993, pp. 26–30.

[18] H. Li, R. Zhao, and X. Wang, "Highly Efficient Forward and Backward Propagation of Convolutional Neural Networks for Pixelwise Classification," Tech. Rep. arXiv:1412.4526, 2014.

[19] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," in *Proceedings of the International Conference on Learning Representations*. arXiv:1312.6229, 2014.

[20] C. M. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[21] H. Neudecker, "Some Theorems on Matrix Differentiation with Special Reference to Kronecker Matrix Products," *Journal of the American Statistical Association*, vol. 64, no. 327, pp. 953–963, 1969.

[22] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," in *Advances in Neural Information Processing Systems, Deep Learning and Representation Learning Workshop*. arXiv:1410.0759, 2014.

[23] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proceedings of the International Conference on Learning Representations*. arXiv:1409.1556, 2015.

[24] T. D. Sanger, "Optimal Unsupervised Learning in Feedforward Neural Networks," Master's thesis, Massachusetts Institute of Technology, 1989.

[25] L. Bottou and Y. LeCun, "Large Scale Online Learning," in *Advances in Neural Information Processing Systems*, vol. 16, 2004, pp. 217–224.

[26] M. Thom and F. Gritschneder, "Rapid Exact Signal Scanning with Deep Convolutional Neural Networks," Tech. Rep. arXiv:1508.06904, 2016.

[27] S. Taylor, *Optimizing Applications for Multi-Core Processors, Using the Intel Integrated Performance Primitives*, 2nd ed. Intel Press, 2007.

[28] L. Dagum and R. Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming," *IEEE Computational Science & Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

Markus Thom, photograph and biography not available at the time of publication.

Franz Gritschneder, photograph and biography not available at the time of publication.