Flexible Acceleration of Convolutions on FPGAs: planning NEURAghe 2.0

Marco Carreras[1], Gianfranco Deriu[1], Paolo Meloni[1]

Università degli Studi di Cagliari, DIEE
marco.carreras@unica.it
gianfranco.deriu@unica.it
paolo.meloni@unica.it

Abstract. Convolutional Neural Networks are commonly employed in applications involving Computer Vision tasks like image/video classification/recognition/segmentation. The increasing focus of the community on this topic, has generated a wide scope of approaches that use different kernel shapes and techniques for executing convolutions with respect to the classic one, such as for example separable convolutions, deformable convolutions or deconvolutions ([4, 5]), frequently used in semantic segmentation tasks ([23, 13]). While it is common knowledge that FPGAs can be used to accelerate classic Convolutional layers in CNNs, there is limited literature about FPGA-based accelerators supporting less regular and common processing kernels ([20]). In our research, starting from the previous experience acquired developing NEURAghe, we plan to improve flexibility of CNN accelerators and to study new methodologies to improve efficiency on the previously mentioned use-cases. As a first experiment we focus on layered approaches based on 1D convolutions, that, as indicated by several recent research results, can be effectively used to classify and segment time series and sequences, as well as in tasks involving sequence modeling. In multiple scenarios a convolution approach applied on the time dimension, hereafter called Temporal Convolution Network (TCN) can outperform classic strategies relying on recurrent networks in terms of accuracy and training time. We modified NEURAghe to support TCN and validate results on an ECG-classification benchmark, achieving up to 95% efficiency in terms of GOPS/s with respect to the accelerator peak performance.

Keywords: Temporal Convolutional Neural Network, TCN, hardware accelerator, FPGA

1 Introduction

Recent studies demonstrate the effectiveness of CNNs, already extensively used for computer vision applications, over tasks like audio synthesis ([26]) and word-level language modeling ([6]). Moreover, among classical convolution approaches, there is an increasing interest towards non regular kernel shapes, like those applied for separable and deformable convolutions. A specific research work, like Bai et al. ([2]), demonstrates that the implementation of a Temporal Convolutional Neural Network over typical sequence modeling tasks can outperform more commonly used Recurrent Neural Networks (RNN).

Nowadays also, a broad range of mono-dimensional CNNs are used for human signals analysis tasks like ECG classification ([8]) or action detection ([16]).

The ubiquitous success of CNNs and their high demands in terms of computing power have motivated during past years a huge outflow of research aimed at developing hardware accelerators for CNN inference. Among other solution, one of the most adopted has been exploiting the cooperation between general purpose processors and FPGAs in modern SoCs, like Xilinx Zynq, providing an efficient implementation of MAC operations on the large amount of DSP Slices available.

Previous mentioned areas of application together with the different convolutional schemes suggest that these kind of hardware accelerators need to be as flexible as possible, supporting multiple CNN's features.

Looking to this objective, and motivated by recent claims about the effectiveness of TCNs in different application domains, this work explores the capabilities of a CNN inference accelerator, NEURAghe [21], over a TCN use-case. To this aim, the architecture has been enhanced to support freely selectable *kernel sizes* and *dilated* convolutions, with freely selectable *dilation rates*, providing the flexibility needed in most TCN algorithms, as well as in regular CNNs.

We report a performance analysis and propose an optimization method relying on *batch processing* to improve efficiency.

The TCN under test with its parameters variability over layers represents a good benchmark for the architecture's flexibility capabilities.

2 State of the art

Convolutional Neural Networks have become the state of the art solutions in fields which concern computer vision tasks like image recognition [11, 18, 1], face detection [25], video classification [15].

They are also used in applications requiring a mono-dimensional elaboration of data like sentence classification [17], speech recognition [10], text understanding [28] and Natural Language Processing tasks [24].

More recent applications involves machine translation [14], audio synthesis [26] and language modeling [6].

On the other hand Recurrent Neural Network (RNN) are considered the go-to solution for sequence modeling tasks [7], although they are difficult to train leading to commonly used architectures like LSTM [12] and GRU [3].

Nevertheless, in a recent study, Bai et al. ([2]) questioned the common association between RNN and sequence modeling. They proposed a Temporal Convolutional Network template that ouperforms recurrent architectures like LSTM and GRU in sequence modeling benchmarks often used to evaluate RNNs.

Another type of application, recently targeted by TCNs, involves human signal analysis like ECG classification ([8],[19]) or action detection ([16], [30], [29]).

With respect to CNN applications for Computer Vision, during years, many hardware solutions have been proposed in order to accelerate the inference task. Among different solutions adopted, one common approach exploits modern SoCs integrating both a general purpose processor and a programmable logic [22, 27].

Other works proposed FPGA based accelerators for LSTM RNN, like [9].

To our knowledge there are no FPGA-based architectures that specifically tackled the problem of hardware acceleration for sequences processed by Temporal Convolutional Neural Networks.

3 TCN model

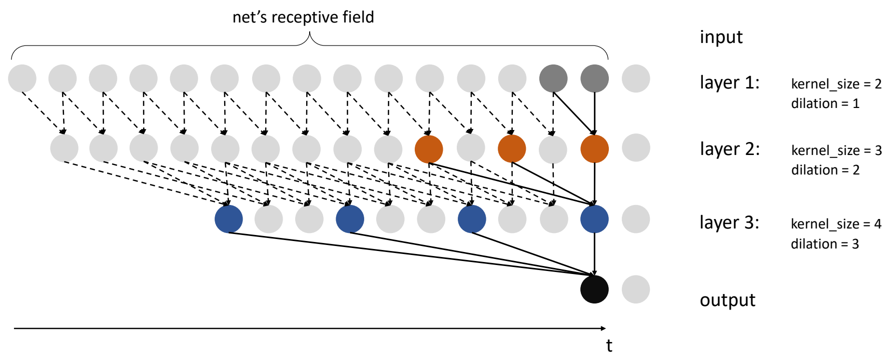

Fig. 1. TCN execution

The input of the network is a continuous sequence of samples from a set of source channels. While in some use-cases input processing can be executed off-line, multiple applications require continuous and near-real-time analysis of the input aimed at the identification of specific events and/or at promptly taking decision on specific actions. In this case, the TCN must analyze as soon as possible any new input sequence sampled by the system and update at every new sample time. At every time step the network processes a sliding window whose minimal size is known as *receptive field* of the network. This is the smallest amount of samples needed to produce an output and depends on convolutional layer parameters such as the *kernel_size* and the *dilation*:

$$receptivefield = 1 + \sum_{l=1}^{L} [kernel\_size(l) - 1] \times dilation(l) \tag{1}$$

where $l \in 1, 2...L$ is a layer of the network.

Figure 1 represents a computational step of a TCN with different layer parameters. Each input or output dot must be considered as a vector with its own dimension in terms of number of channels. The figure shows how many input samples are needed by each layer to perform a valuable convolution and how *kernel_size* and *dilation* affect the *receptive field*. Moreover, especially with different dilation rates, the network can have longer memory without increasing too much the depth and the number of parameters to train.

4 NEURAghe Architecture

The starting point for this NEURAghe architectural template has been the one described in Meloni et al. [21]. This architecture exploits the cooperation between the ARM Cortex-A9 processing system and the programmable logic in Xilinx Zynq devices. Communication at the PS-PL interface is guaranteed by the high performance 64 bit ports and two general purpose 32 bit ports.

The programmable logic hosts the Convolution Specific Processor (CSP) while the processing system acts as a General Purpose Processor (GPP) dealing with tasks hardly to accelerate in the programmable logic, like fully connected layers execution. In this work, the CSP has been enhanced with respect to the previous

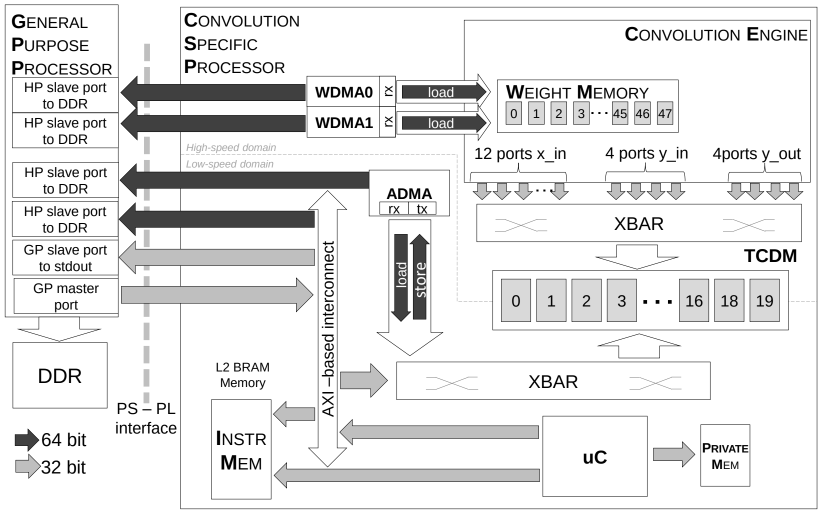

Fig. 2. NEURAghe 2.0 architecture

version to improve the accelerator's flexibility towards various networks characteristics. In particular there has been a substantial modification of the Convolution Engine, previously characterized by the so called *Line Buffer* and a complete different SoP module model. The former, in charge to supply convolutional windows to the SoP matrix, has been removed thanks to the new pixel fetching method, while the latter, composed by a double trellis of pipelined DSPs, has been completely redesign. Both changes let the architecture to be more flexible. Finally the transfers capabilities are improved by doubling the Weight DMA.

4.1 Convolution Engine organization

The Convolution Engine is the computational core of the accelerator. It is designed to execute a high number of Multiply and Accumulate (MAC) operations in parallel, to relief the host processor from the most computational-intensive tasks of a CNN. It is composed by a matrix of M columns by N rows of Sum of Product (SoP) units in charge to calculate the contribution of M input features to N output features. Partial result from SoPs in each row are summed together by means of N Shift Adder modules.

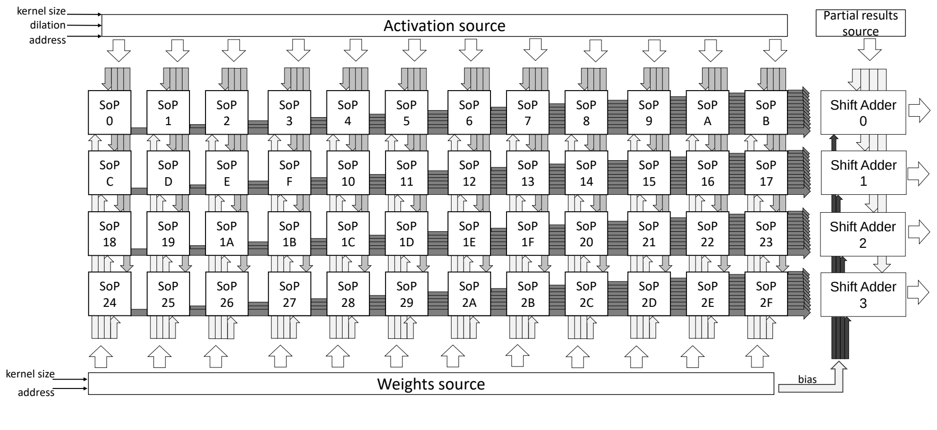

Fig. 3. Convolution Engine

In more detail, each SoP reads 4 *samples/cycle* and executes 4 $MACs/cycle$, one for each of the 4 consecutive kernel windows applied to an IF. In this way a SoP module produces 4 new output samples after *kernel size* cycles. Every output sample produced can be sent to the Shift Adder module.

In this configuration the C.E. has a 12×4 SoP matrix, therefore, every clock cycle, 4 samples of 12 input features are fetched to SoPs from the C.E. internal memory (TCDM), by means of the Activation Source modules. Samples of every IF are sent to the 4 SoPs of a column. Whereas 12×4 weight kernels are independently fetched from Weight Memory banks by means of the Weights source module.

SoP modules are built using 4 Xilinx DSP48E1 primitives configured to perform a MAC operation per cycle using the internal loop to iterate among consecutive partial result given by a weight kernel application. This design allows the accelerator:

– to be *kernel size* agnostic,
– to execute convolutions with multiple stride values without performance overhead.

As the Convolution Engine works on 16-bit sample data, every SoP reads 64-bit activation data per cycle and multiplies all four pixels for the same kernel weight. Thus each SoP needs to read only 16-bit weight data per cycle.

Shift Adder modules read 64-bit data that are 4 16-bit values resulting from the previous 12 IF contribution and produce 64-bit data output summing together these inputs and those given by the actual 12 IF contribution.

4.2 Scheduling

As NEURAghe exploits a double buffered memory policy, local memory banks are subdivided in two parts, one dedicated to the data needed for the actual computational step, while the other allows to perform data transfers that will be used for the next phase. This allows the accelerator to overlap transfers with computational phases aiming to reduce idles.

From the efficiency point of view, the best situation is when all transfers are overlapped by execution phases because it is exploited all the computational power of the accelerator.

4.3 TCN model on NEURAghe

Temporal CNN characteristics allow NEURAghe to handle samples of different layers in a specific way that slightly differs from regular convolutions. As mentioned above, *kernel size* and *dilation* affect the *receptive field* of the network that defines the number of input samples that must be processed to gain a new output. Furthermore, for every layer a specific *receptive field* can be considered, given by:

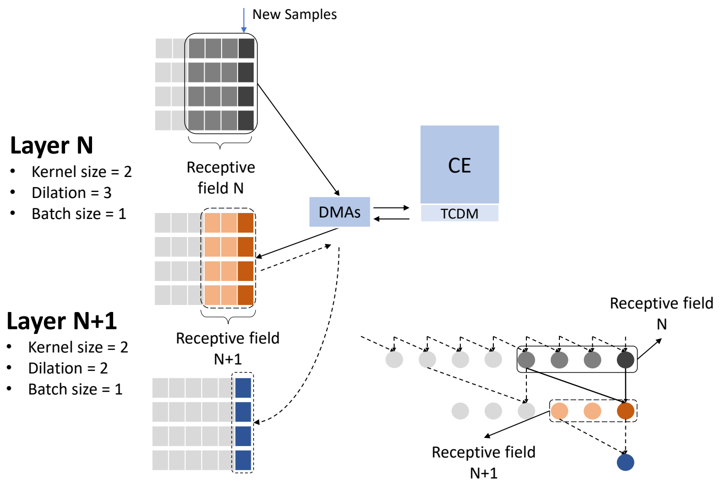

Fig. 4. TCN execution example on NEURAghe. Receptive field samples are loaded to C.E. memory for execution and only new ones are stored back in memory.

$$1 + (kernel\_size - 1) \times dilation \tag{2}$$

that is the minimum number of input samples per channel needed to obtain an output sample from a layer. Figure 4 shows the memory status in different moments for two layers of an example network. In this example, layers N and $N+1$ are characterized by the same *kernel size* of 2 and *dilation* of 3 and 2 respectively. For the sake of simplicity, it is represented a single sample update instead of the 4 processed by the accelerator.

As it can be seen, for every computational step, it is necessary to retain in memory only the samples equal to the specific *receptive field* of each layer. As a consequence, every transfer concerns only this minimum amount of samples per channel.

4.4 Improving through batch processing

The previously mentioned approach minimizes the classification/recognition latency. It produces outputs as soon as possible and repeats network execution every time that a new sample is available to update the input sliding window.

However, this can determine the performance of the system to be easily bandwidth limited. All the network parameters/weights have to be loaded on the accelerator local memory and are used only for the production of one single output. This decreases significantly the *operational intensity* of the application.

As an example we show a roofline model of our system in Figure 5. The leftmost

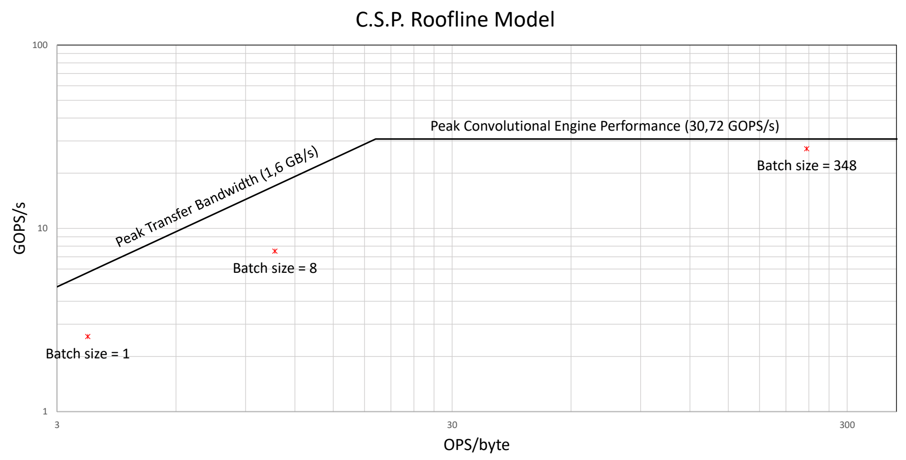

Fig. 5. Use case benchmark's Roofline Model for the Convolution Specific Processor with respect to different batch size

red symbol indicates the performance achieved when using the sample-by-sample processing on the use-case that will be presented in the following.

As it may be noticed, in this case, the system performance are definitely limited by input/output bandwidth.

If the application allows to trade-off some increase in latency to improve performance, a solution to this problem may rely on batch processing. We can pre-buffer input samples and process longer sample sequences producing more outputs with

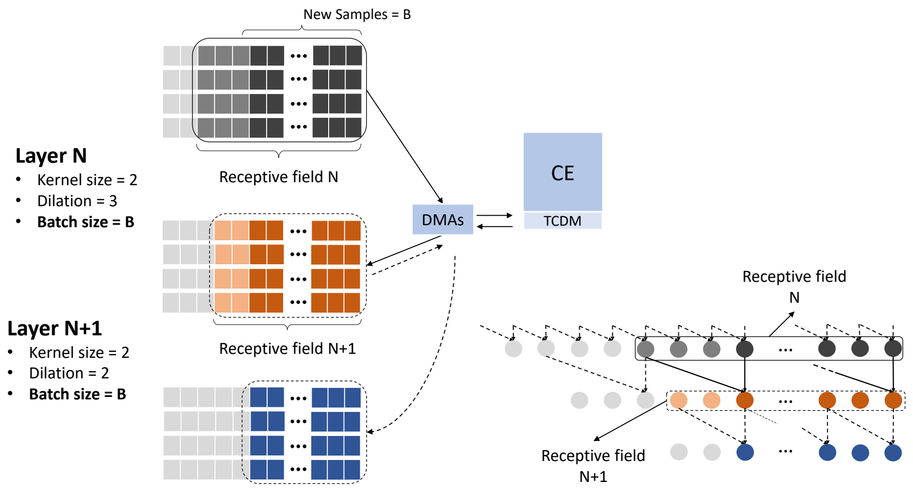

Fig. 6. TCN execution with batching on NEURAghe The number of samples needed depends on the receptive field but increased of *Batch size* = *B* samples. The C.E. produces B new samples.

every TCN execution. Figure 6 shows the transfer scheduling when *batch size* is increased.

Increasing the *batch size*, with the same weight transfers from DDR to local memory, we perform more computations. In this way, it is possible to increase the operational complexity and to gain efficiency, see other red symbols in Figure 5.

4.5 Resources utilization on target board

The NEURAghe architecture is scalable and can be implemented in different devices like those which are part to the Xilinx Zynq-7000 SoC family. In particular, the configuration described above is highly suitable for boards, like the Zedboard, integrating a Xilinx Zynq Z-7020. Table 1 shows the resource occupation of the reconfigurable logic of the device.

Table 1. Resource occupation on a Xilinx Zynq Z-7020

	DSP	BRAM	LUTs (logic)	LUTs SR	Regs
Used	192	120	47230	259	26942
Available	220	140	53200	53200	106400
%	87.27	85.71	88.78	1.49	25.32

It is worth noticing that the architecture uses 192 out of the 220 DSP blocks available in the device, so the processing power utilization is very high. Also the

Block RAM primitives are extensively used due to the particular Weights Memory implementation. The accelerator on Zedboard is clocked at 80 MHz.

5 Use case network: ECG Classification

The use case that we have chosen as benchmark is a network for ECG monitoring and classification (Goodfellow et al. [8]). It performs a classification over single lead ECG waveforms as either Normal Sinus Rhythm, Atrial Fibrillation, or Other Rhythm and reaches around 90% average accuracy over targeted single lead ECG waveforms. Experimental dataset for this network are characterized by 16 bit batches of data sampled at 300 Hz frequency. The network consists of 13 computational blocks mostly made of a 1D Convolutional layer, a batch normalization layer, a ReLU and a dropout stage. Only 3 layer have also a Max Pooling stage with a pooling size of 2 between ReLU and Dropout. Computational blocks have decreasing *kernel sizes* while *dilation* parameter increases through the network. The analysis of the architecture performance has been made distinguishing three operative modes: *latency constrained network*, *latency unconstrained network*, *real-time execution*. For the first one the minimum input sized version of the network has been considered since it ensures best performance in terms of latency.

In Table 2 are shown the characteristics of convolutional layers of the network along with minimum input size required. The *latency constrained* configuration

Table 2. Convolutional Layer characteristics and minimum input sizes required by the CE

	input size	IF	OF	*kernel size*	*dilation rate*
Type 1	27	1	320	24	1
Type 2	34	320	256	16	2
Type 3	64	256	256	16	4
Type 4	32	256	128	8	4
Type 5	46	128	128	8	6
Type 6	60	128	128	8	8
Type 7	60	128	64	8	8
Type 8	60	64	64	8	8

provides best performance from the point of view of total elaboration time for a new output value of the network. On the other hand this configuration lacks in terms of efficiency with respect to peak $GOPS/s$ reachable by NEURAghe (first column of Table 3).

Low efficiency for this configuration is due to the huge weight transfer overhead with respect to the very short activation load time and execution time. So, despite the double buffered scheduling strategy, transfer and computation phases hardly overlap.

If the target application has no latency constraint it is possible to work with much more activation samples as input for every layer. In particular when the execution time reaches and surpasses transfer times the architecture can get performances very close to the peak, as it is shown in third column of Table 3.

Table 3. Convolutional Layer performance for *latency constrained*, *real-time execution* and *latency unconstrained* network

	Batch Size = 1			*Batch Size* = 8			*Batch Size* = 348		
	exec. time [*ms*]	*GOPs/s*	*effic.*	*exec. time* [*ms*]	*GOPs/s*	*effic.*	*exec. time* [*ms*]	*GOPs/s*	*effic.*
Type 1	0.413	0.148	0.0048	0.467	0.394	0.013	2.484	2.176	0.07
Type 2	3.172	3.305	0.107	3.256	9.66	0.314	31.409	29.37	0.95
Type 3	2.64	3.17	0.103	2.755	9.13	0.297	25.644	28.78	0.937
Type 4	0.87	2.4	0.078	0.874	7.19	0.234	6.836	26.99	0.878
Type 5	0.545	1.92	0.062	0.558	5.64	0.183	3.6	25.63	0.834
Type 6	0.543	1.93	0.062	0.558	5.64	0.183	3.61	25.56	0.832
Type 7	0.38	1.38	0.044	0.385	4.08	0.132	1.96	23.54	0.766
Type 8	0.29	0.88	0.029	0.304	2.58	0.084	1.236	18.66	0.607

As a third case it can be considered that for which the latency constraint is not as tightening as for the first case and it is possible to increase the efficiency without compromise performance from the point of view of the execution time.

In particular it is possible to do interesting assumption about a real time operating mode by referring to the sample rate mentioned in the paper, that is 300 Hz.

Second column of table 3 also shows that by buffering a small amount of samples, that is increasing the *batch size* parameter by 8 *samples*, before feeding the accelerator, it is possible to substantially gain in efficiency without loss in performance with respect to the *latency constrained* configuration. Moreover, having an initial buffering of 8 *samples* means that the accelerator has a new batch every 26.67 ms which is enough to complete an end-to-end computation for this network.

6 Conclusions and future work

In this work has been presented an application towards Temporal Convolutional Networks of NEURAghe, an FPGA-based hardware accelerator enhanced to be *kernel* and *dilation rate* agnostic and also to process inputs with multiple stride values, without overhead. Motivated by recent claim about TCNs implementation over various applications it has been made an explorations of the architecture's performances with respect to convolutional layers of an use case TCN. Results showed that the architecture has a good flexibility over various layer characteristics. It has also been showed how performances improve by changing the computational paradigm, going from a *latency constrained* approach to a *batched* approach, by agreeing with a certain latency.

The next step can be exploring different architectural configurations suitable both for different target devices and different TCNs in order to find the solution that adapt best.

References

1. Deep image: Scaling up image recognition. CoRR **abs/1501.02876** (2015), http://arxiv.org/abs/1501.02876, withdrawn.
2. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 (2018)
3. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259 (2014)
4. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. CoRR **abs/1610.02357** (2016), http://arxiv.org/abs/1610.02357
5. Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y.: Deformable convolutional networks. CoRR **abs/1703.06211** (2017), http://arxiv.org/abs/1703.06211
6. Dauphin, Y.N., Fan, A., Auli, M., Grangier, D.: Language modeling with gated convolutional networks. CoRR **abs/1612.08083** (2016), http://arxiv.org/abs/1612.08083
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
8. Goodfellow, S., Goodwin, A., Eytan, D., Greer, R., Mazwi, M., Laussen, P.: Towards understanding ecg rhythm classification using convolutional neural networks and attention mappings (08 2018)
9. Guan, Y., Yuan, Z., Sun, G., Cong, J.: Fpga-based accelerator for long short-term memory recurrent neural networks. In: 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). pp. 629–634 (2017). https://doi.org/10.1109/ASPDAC.2017.7858394
10. Hannun, A.Y., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., Ng, A.Y.: Deep speech: Scaling up end-to-end speech recognition. CoRR **abs/1412.5567** (2014), http://arxiv.org/abs/1412.5567
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), http://arxiv.org/abs/1512.03385
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
13. Jégou, S., Drozdzal, M., Vázquez, D., Romero, A., Bengio, Y.: The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. CoRR **abs/1611.09326** (2016), http://arxiv.org/abs/1611.09326
14. Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A.v.d., Graves, A., Kavukcuoglu, K.: Neural machine translation in linear time. arXiv preprint arXiv:1610.10099 (2016)
15. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 1725–1732 (2014)
16. Kim, T.S., Reiter, A.: Interpretable 3d human action analysis with temporal convolutional networks. CoRR **abs/1704.04516** (2017), http://arxiv.org/abs/1704.04516
17. Kim, Y.: Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. pp. 1097–1105. NIPS'12, Curran Associates Inc., USA (2012), http://dl.acm.org/citation.cfm?id=2999134.2999257
19. Li, D., Zhang, J., Zhang, Q., Wei, X.: Classification of ecg signals based on 1d convolution neural network. In: 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom). pp. 1–6 (10 2017). https://doi.org/10.1109/HealthCom.2017.8210784

20. Liu, B., Zou, D., Feng, L., Feng, S., Fu, P., Li, J.: An fpga-based cnn accelerator integrating depthwise separable convolution. Electronics **8**(3), 281 (2019)
21. Meloni, P., Capotondi, A., Deriu, G., Brian, M., Conti, F., Rossi, D., Raffo, L., Benini, L.: Neuraghe: Exploiting CPU-FPGA synergies for efficient and flexible CNN inference acceleration on zynq socs. CoRR **abs/1712.00994** (2017), http://arxiv.org/abs/1712.00994
22. Mittal, S.: A survey of fpga-based accelerators for convolutional neural networks. Neural computing and applications pp. 1–31 (2018)
23. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. CoRR **abs/1505.04597** (2015), http://arxiv.org/abs/1505.04597
24. Santos, C.D., Zadrozny, B.: Learning character-level representations for part-of-speech tagging. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14). pp. 1818–1826 (2014)
25. Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1701–1708 (2014)
26. Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A.W., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. SSW **125** (2016)
27. Wang, C., Gong, L., Yu, Q., Li, X., Xie, Y., Zhou, X.: Dlau: A scalable deep learning accelerator unit on fpga. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **36**(3), 513–517 (2016)
28. Weston, J.E.: Dialog-based language learning. In: Advances in Neural Information Processing Systems. pp. 829–837 (2016)
29. Yang, J.B., Nguyen, M.N., San, P.P., Li, X.L., Krishnaswamy, S.: Deep convolutional neural networks on multichannel time series for human activity recognition. In: Proceedings of the 24th International Conference on Artificial Intelligence. pp. 3995–4001. IJCAI'15, AAAI Press (2015), http://dl.acm.org/citation.cfm?id=2832747.2832806
30. Zeng, M., Nguyen, L.T., Yu, B., Mengshoel, O.J., Zhu, J., Wu, P., Zhang, J.: Convolutional neural networks for human activity recognition using mobile sensors. In: 6th International Conference on Mobile Computing, Applications and Services. pp. 197–205 (11 2014). https://doi.org/10.4108/icst.mobicase.2014.257786