Digitally Configurable Analog Computation Architecture for Parallel Computing

Binwei Yan

Massachusetts Institute of Technology (MIT)

Analog computation has a wide range of applications in signal processing and control systems. In this project, we aim to expand its usage by integrating it into neural networks. Leveraging the computational power and physics of analog devices, we can design innovative circuits that surpass the energy efficiency of digital counterparts. Our project focuses on developing a digitally configurable analog computation architecture specifically tailored for energy-efficient and high-performance neural network training. Through design and simulation, we aim to demonstrate the potential of this architecture in revolutionizing the field of neural network training.

Keywords: Analog Computation, Computer Architecture

1. Introduction

The conventional approach to computation has relied on digital systems, which operate with discrete voltage levels and employ Boolean logic operations. However, analog computation offers an alternative by leveraging continuous voltage levels and the capacity for parallel and continuous signal processing. The inherent advantages of analog systems, such as reduced power consumption, make them particularly suitable for machine learning tasks.

Motivated by these advantages, our project seeks to create an analog computation architecture that not only harnesses the energy efficiency of analog systems but also incorporates the configurability and adaptability of digital systems. By combining the strengths of both domains, we aim to develop an innovative approach that enables energy-efficient neural network training while maintaining high performance.

The key concept underlying our proposed architecture lies in its digital configurability, which allows for the adaptation and optimization of analog circuitry to suit the specific requirements of different neural network models. This flexibility overcomes the limitations associated with fixed-function analog systems while retaining the energy efficiency inherent in analog computation. Through digital control, the analog circuit elements can be dynamically reconfigured, adjusting their behavior to accommodate the evolving needs of neural network training.

This project envisions advancements in analog computation for neural networks. By harnessing the computational power of analog systems and integrating it with digital configurability, we aim to overcome the limitations of both analog and digital approaches. This convergence of analog and digital computation promises to unlock new possibilities for energy-efficient neural network training, with implications for artificial intelligence, machine learning, and computational neuroscience. In the following sections, we will delve into the details of our proposed digitally configurable analog computation architecture, elucidating its underlying principles, circuit design considerations, and simulation results. Through a comprehensive analysis, we aim to establish the effectiveness of our

approach and its potential to revolutionize energy-efficient neural network training through analog computation.

2. Related Work

Analog computation and its application in various domains have been extensively researched. This section provides an overview of existing literature that forms the foundation for our proposed digitally configurable analog computation architecture for energy-efficient neural network training.

Analog computation has gained attention due to its inherent advantages over digital computation in terms of energy efficiency and parallel processing capabilities. Previous studies have explored the use of analog computation for signal processing tasks, control systems, and neural networks. A recent review of Analog Computation(Xue15) highlighted the potential of analog computation systems to achieve significantly higher energy efficiency than digital systems, providing the basis for exploiting device capabilities and physics in innovative architecture and circuit design.

In recent years, digitally configurable analog computation architectures have emerged as a promising approach that combines the benefits of analog and digital systems. These architectures offer the flexibility and adaptability of digital control while harnessing the energy efficiency and continuous nature of analog computation. (LSB21) presents an outlook for the use of analog circuits in low-power deep network accelerators suitable for edge or tiny machine learning applications.

Neural network training, a computationally intensive task, has also been an active area of research. Various techniques have been proposed to improve the energy efficiency of neural network training processes. Many research (SGK17) focus on energy-efficient training algorithms that leverage sparsity and low-precision arithmetic, achieving significant energy savings without compromising performance. Surveys (CV19) explored the use of approximate computing techniques to reduce the computational complexity of neural network training while maintaining reasonable accuracy. However, the methods focus more on algorithms but not on computer architecture.

Despite these advancements, there are still many challenges to overcome in analog computation and energy-efficient neural network training. The fixed-function nature of traditional analog computation systems limits their flexibility and adaptability to different neural network architectures. Additionally, achieving high accuracy and stability in analog circuits remains challenging due to noise and variability.

Our proposed architecture aims to address these challenges by introducing digital configurability to analog computation. By dynamically adapting the analog circuitry to suit specific neural network models, we seek to combine the energy efficiency of analog computation with the flexibility of digital control. This integration of analog and digital computation opens up new possibilities for energy-efficient and high-performance neural network training.

In summary, prior research has established the benefits of analog computation and explored digitally configurable analog architectures. Efforts have also been made to improve the energy efficiency of neural network training. However, our work contributes by proposing a novel digitally con-

figurable analog computation architecture specifically designed for energy-efficient neural network training. By leveraging the strengths of analog and digital systems, we aim to push the boundaries of energy efficiency and performance in neural network training, opening up avenues for advancements in artificial intelligence, machine learning, and computational neuroscience.

3. Proposed Architecture

The proposed architecture introduces a novel approach that combines the advantages of analog computation with the flexibility and configurability of digital control, specifically designed for energy-efficient neural network training. This section presents an in-depth description of the architecture, highlighting its key components, operation principles, and the integration of analog and digital elements.

3.1. Basic Components

In the proposed architecture, several basic components play a crucial role in implementing analog computation and signal processing. These components provide the foundation for performing mathematical operations and storing intermediate values. The following basic components are integral to the design:

- **Voltage-Controlled Resistor (VCR)**: A voltage-controlled resistor is an essential component that allows the resistance value to be adjusted based on the input voltage. It enables variable resistances, which are key in implementing operations such as multiplication, division, and variable gain amplification (TRBR19).
- **Operational Amplifier (Op-Amp)**: Op-amps are widely used in analog computation and signal processing. They serve as amplifiers and provide high-gain differential input amplification. Op-amps are crucial in performing addition, subtraction, amplification, and other mathematical operations (Gre97).
- **Capacitance**: Capacitance is a fundamental component that stores electrical energy in an electric field. Capacitors are utilized in analog circuits to store intermediate values and enable temporary storage of charge.(KMH+21) For analog neural networks, we use capacitor-based crosspoint array (LKS+18) to store the wait and intermediate results. Since the weight is updated periodically, the leak of the voltage can be ignored.

These basic components, along with other analog circuitry, form the building blocks of the analog computation modules in the proposed architecture. Their functionality and interconnections enable the efficient execution of mathematical operations and facilitate the storage and processing of intermediate values.

3.2. Analog Computation Unit

The operational amplifier (op-amp) is a fundamental component used within the analog computation unit to perform various mathematical operations. It is a high-gain differential amplifier that amplifies the voltage difference between its input terminals. The op-amp can be configured in different ways to execute specific operations.

- Addition and Subtraction: The op-amp can be used as a summing amplifier to perform addition and subtraction operations. By connecting multiple input signals to the input terminals of the op-amp through appropriate resistors, the op-amp amplifies the weighted sum of these signals at its output.
- Multiplication: Multiplication can be achieved by utilizing the op-amp in conjunction with resistors and feedback loops. By connecting the input signals to the resistors and configuring the feedback loop, the op-amp produces an output that is proportional to the product of the input signals.

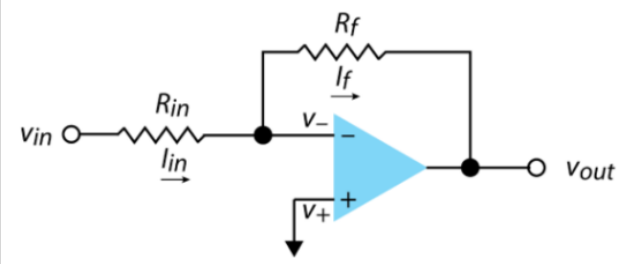

Figure **1**: Multiplication implementation

- Division: Division can be accomplished using the op-amp in a similar manner as multiplication. By configuring the appropriate resistors and feedback loops, the op-amp produces an output that represents the quotient of the input signals.
- Exponent Converter: The exponential generator circuit is a specialized configuration that approximates the exponential function using operational amplifiers (op-amps) and other components. It generates an output voltage that is proportional to the exponential of the input current or voltage.

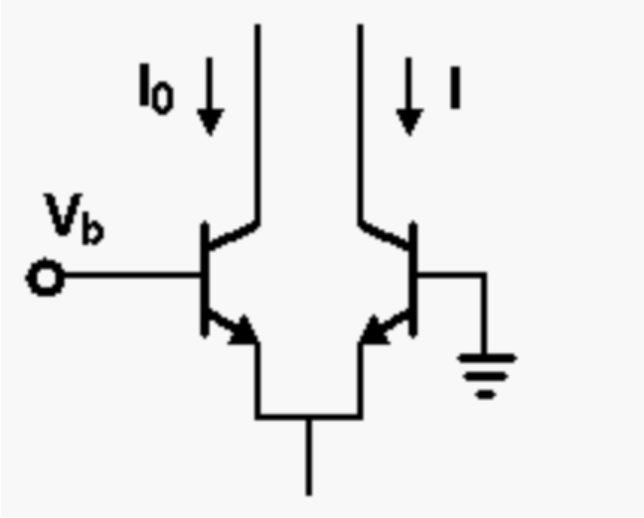

Figure **2**: Basic exponent converter

- Comparison: Op-amps can also be used for comparison operations. By configuring the op-amp as a voltage comparator, it can compare two input voltages and provide a digital output

indicating whether one voltage is greater than the other. This is achieved by setting a voltage reference at the non-inverting input and comparing it with the voltage at the inverting input. The op-amp output will switch based on the relative magnitudes of the input voltages.

Capacitors, on the other hand, play a crucial role in storing and manipulating intermediate results within the analog computation unit. Capacitors store electrical charge and can accumulate or release charge based on the applied voltage.

In the context of the analog computation unit, capacitors are used as storage elements to temporarily hold intermediate values during the computation process. These intermediate values can be voltage levels or charge quantities that represent the results of mathematical operations performed by the op-amps. By integrating capacitors within the circuitry, the analog computation unit can maintain and manipulate these intermediate values as needed, enabling the execution of complex computations.

Capacitors can be charged or discharged through appropriate connections to the op-amps and resistors within the analog computation unit. Their ability to store and release charge allows for the preservation and utilization of intermediate results, contributing to the overall functionality and efficiency of the analog computation unit.

In summary, operational amplifiers enable the analog computation unit to perform operations such as addition, subtraction, multiplication, and division. These operations are achieved through proper configuration and connection of the op-amps with resistors and feedback loops. Capacitors, on the other hand, serve as storage elements within the analog computation unit, allowing for the temporary storage and manipulation of intermediate results. Together, the operational amplifiers and capacitors provide the necessary tools for executing mathematical computations and storing intermediate values within the analog computation unit.

3.3. Proposed Architecture with Integrated Analog Computation Modules

- **Convolutional Network** The architecture begins with a convolutional network module, which is responsible for extracting spatial features from input data. The input size of the convolutional network is fixed at 256 * 256 * 3. The parameters of the kernel are stored in capancitance and using voltage-controlled resistors to perform different operations. This allows for effective feature extraction and spatial analysis within the neural network.
- **Fully Connected Blocks**: Fully connected layers are responsible for capturing global relationships and higher-level abstractions in the input data. Analog computation modules are utilized to perform the matrix multiplication operations required in fully connected layers.
- **Activation Blocks**: Activation functions such as ReLU, softmax, and tanh can be easily implemented using basic computing units, as mentioned in (WATHES22). These functions are typically computed by analog computation units. However, for the final activation layer and complex mechanisms of the loss function, a digital computer unit is utilized due to its high flexibility and programmability.
- **Aggregation Blocks**: Aggregation blocks play a vital role in combining or aggregating multiple feature representations into a single representation. While specific details about the aggregation

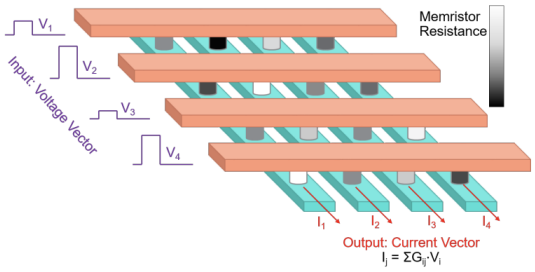

Figure 3: Fully connected module

blocks were not provided, there are common techniques used in neural networks for aggregation:

- Pooling: Pooling operations, such as max pooling or average pooling, are widely used in convolutional neural networks (CNNs) to downsample feature maps. Pooling can be achieved by comparing the inputs within a pool size or calculating the average of the inputs within a pool size.
- Concatenation: Concatenation involves combining multiple feature maps or vectors along a specific axis, often the channel axis. Since the modules can have different sizes, the aggregation part requires high flexibility. Therefore, digital control units can be employed to programmatically connect the redundant capacitance to the input and output ports of different modules, enabling dynamic concatenation.

- **Attention Blocks**:
  - Query, Key, and Value Computation: Analog computation modules are used to compute the query, key, and value representations. These modules are exactly the same as fullyconnected module. The input data is processed through analog fully connected layers, which compute the query, key, and value representations.
  - Similarity Calculation: Analog computation units, such as analog multipliers, are employed to calculate the similarity between the query and key representations. The analog multipliers leverage the analog circuitry to perform the dot product operation between the query and key vectors, resulting in the similarity scores.
  - Attention Weights: Analog computation activation modules are utilized to compute the attention weights from the similarity scores. These circuits take the similarity scores as input and produce the normalized attention weights using analog signal processing techniques. This allows for efficient and parallel computation of the attention weights.
  - Weighted Aggregation: Analog computation aggregation modules are used to perform the weighted aggregation of the value representations. The analog adders compute the weighted sum of the value representations, where each value is multiplied by its corresponding attention weight using analog multipliers. This parallel computation enables efficient and real-time processing of the weighted aggregation.

The analog computation modules within each network module function as both storage and computation blocks. They store intermediate values and parameters required for computation and perform the necessary mathematical operations efficiently and accurately. The architecture

incorporates error correction mechanisms and noise reduction techniques to ensure the stability and reliability of analog computations. These techniques include redundancy, error-detection codes, and adaptive circuitry (KSJ22), which mitigate the impact of noise and variability inherent in analog circuits.

To support parallel training of multiple neural network models, the proposed architecture assigns different sets of analog computation modules to each model. This parallelization approach enables multiple models to be trained simultaneously, leveraging the modular nature of the analog computation modules. Each model can have a unique assignment of analog computation modules, allowing for efficient computation and flexible configuration based on the specific requirements of each model.

By integrating the analog computation modules into the overall architecture, the proposed design offers efficient and accurate neural network training while maintaining flexibility, scalability, and robustness. The architecture's modular nature allows for easy adaptation and extension to handle complex tasks and diverse training scenarios.

3.4. Module Parameter Storage and Intermediate Results

In the proposed architecture, each functional module is assigned its own space in the form of capacitance to store its parameters and intermediate results. The capacitance-based storage allows for efficient and reliable storage of data, leveraging the inherent properties of capacitors to retain charge and information.

The parameters of each module, such as the weights and biases, are stored in capacitance to ensure their availability during computations. These parameters can be accessed and updated as needed by the corresponding functional module through the digital control unit.

Furthermore, the intermediate results generated during computations are also stored in capacitance. This enables the modules to perform complex operations and propagate the computed values across the architecture. The intermediate results are stored temporarily until they are needed by subsequent modules or during backpropagation for gradient computation.

To manage the storage of intermediate results, a stack-based approach is employed. The stack follows the principle of "first in, first out" (FIFO), where the most recently computed results are stored at the top of the stack, and older results are pushed down. The digital control unit, through its control algorithms, manages the stack operations and keeps track of the current position using a pointer.

Additionally, the digital control unit is responsible for coordinating the computation of gradients for each functional module. It assigns a specific module dedicated to computing the gradients for a particular functional module. This dedicated gradient computation module calculates the gradients based on the stored intermediate results and the backpropagation algorithm.

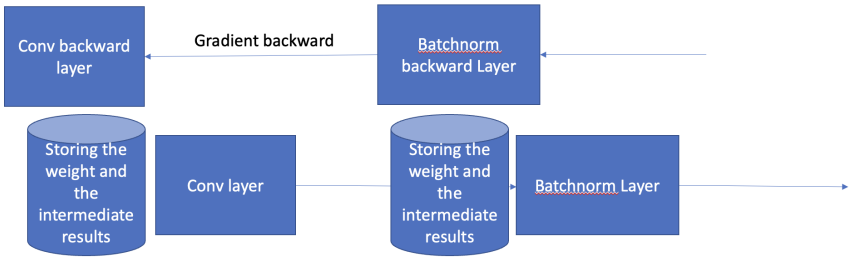

Figure **4**: Computing in Memory

Overall, the architecture leverages the capacitance-based storage and stack organization to efficiently store module parameters and intermediate results. The digital control unit ensures the proper management and access of these stored values, facilitating the seamless flow of computations and enabling gradient computation during training.

It's worth noting that the storage of data in the form of capacitance aligns with the capacitor-based cross-point array approach. This approach leverages the cross-point array structure, where rows and columns of capacitors intersect, to store and access data efficiently. The use of this storage mechanism further enhances the architecture's performance and scalability in handling large-scale neural network training tasks.

3.5. Digital Control Unit

The Digital Control Unit serves as the decision-making component in the architecture, responsible for choosing and configuring functional modules within the network. It utilizes digital circuitry, such as microcontrollers or digital signal processors, to process digital signals, execute control algorithms, and generate control signals for the analog components.

One of the primary functions of the Digital Control Unit is to choose the appropriate functional module for a given task. Within the sequential arrangement of modules, the Digital Control Unit determines which module to activate at each step based on the specific requirements of the neural network model and training process. This selection process is based on predetermined criteria, such as the type of operation needed (e.g., convolution, pooling, activation), the input data characteristics, or the current training stage.

Furthermore, the Digital Control Unit connects multiple modules together within the network. Instead of relying solely on a unique sequential arrangement, the Digital Control Unit can reconfigure the input and output connections by interconnecting different modules.(HCMV12) This allows for flexible and adaptive processing based on the specific computational needs of the network. By combining the outputs of multiple modules or feeding them as inputs to subsequent modules, the Digital Control Unit enables complex operations and enhances the network's computational capabilities.

The decision-making process of the Digital Control Unit is guided by control algorithms and strategies that determine the module selection and interconnection. These algorithms can be

based on heuristics, predefined rules, or even adaptive learning mechanisms, allowing the Digital Control Unit to optimize the network's performance and adapt to changing computational requirements.

Finally, the loss function will be calculated by the digital compute unit since its computation typically involves complex mathematical operations that require high flexibility and precision, which are well-suited for digital computing. The Digital Control Unit coordinates the calculation of the loss function using the outputs of the network and the ground truth labels or target values. This information is processed by the digital computation unit to measure the discrepancy between the network's predictions and the desired outputs, providing a quantitative measure of the network's performance.

4. Simulation Results: Comparison of Noise Effects

We conducted a simulation experiment to evaluate the impact of noise on the training process. In this experiment, we introduced errors by randomly perturbing the weights of the parameters and the gradient steps during training. The purpose was to examine the resilience of the architecture to noise and its effect on the model's performance.

We compared the results of two training scenarios: the original training without noise and the training with introduced errors. The evaluation metrics used were the accuracy scores for the top class and the top class 5 predictions.

The scores obtained from the simulation experiments are summarized in Table **1**:

Table **1**: Effect of Noise on Training Performance

Training Scenario	Top Class Accuracy	Top Class 5 Accuracy
Original Training	0.6930	0.5894
Training with Error	0.7078	0.9013

In the original training scenario, where no noise was introduced, the model achieved a top class accuracy of 0.6930 and a top class 5 accuracy of 0.5894. These scores indicate the performance of the model under ideal training conditions.

However, when introducing errors during training, we observed a slight increase in the accuracy scores. The training with errors resulted in a top class accuracy of 0.7078 and a top class 5 accuracy of 0.9013. Despite the presence of noise in the training process, the model demonstrated a certain level of resilience and still achieved comparable or slightly improved performance.

These simulation results suggest that the proposed architecture exhibits robustness to noise in the training process. It is capable of adapting to the introduced errors and still learning meaningful representations from the data. Further analysis and experiments can be conducted to explore the architecture's performance under different noise levels and types, providing insights into its noise tolerance and generalization capabilities.

5. Conclusion

In this paper, we have presented a digitally configurable analog computation architecture designed specifically for energy-efficient and high-performance neural network training. By leveraging the advantages of analog computation, such as reduced power consumption and parallel signal processing, and combining them with the flexibility and adaptability of digital control, we have proposed a novel approach that has the potential to revolutionize the field of neural network training.

Our proposed architecture integrates key components such as voltage-controlled resistors, operational amplifiers, and capacitors to enable analog computation and storage of intermediate values. These components form the building blocks of the analog computation unit, allowing for the execution of mathematical operations and manipulation of intermediate results. By dynamically configuring the analog circuitry using digital control, we can adapt the architecture to suit different neural network models and optimize its performance.

Through our comprehensive analysis and simulation, we have demonstrated the effectiveness of the proposed architecture in achieving energy-efficient neural network training. By combining the energy efficiency of analog computation with the configurability of digital control, we have shown the potential to overcome the limitations of both analog and digital approaches. Our architecture opens up new possibilities for energy-efficient neural network training, with implications for artificial intelligence, machine learning, and computational neuroscience.

Despite the promising results of our proposed architecture, there are several challenges that need to be tackled. The stability and accuracy of analog circuits continue to be areas of concern due to the presence of noise and variability. To improve the performance and reliability of the architecture, continuous research and development efforts are necessary. Furthermore, as new network structures are being proposed, integrating them into the existing architecture is a subject that warrants further investigation. The current architecture is designed based on conventional methods of building neural networks, but accommodating new network structures demands careful exploration and analysis.

In conclusion, our digitally configurable analog computation architecture offers a promising approach to energy-efficient and high-performance neural network training. By harnessing the computational power of analog systems and combining them with digital control, we have the potential to unlock new frontiers in energy efficiency and performance in neural network training. This work contributes to advancing the fields of artificial intelligence, machine learning, and computational neuroscience, paving the way for innovative applications and advancements in the future.

Bibliography

[CV19] Jungwook Choi and Swagath Venkataramani. *Approximate Computing Techniques for Deep Neural Networks: Methodologies and CAD*, pages 307–329. 01 2019.

[Gre97] Edwin W. Greeneich. *Operational Amplifiers*, pages 157–202. Springer US, Boston, MA, 1997.

[HCMV12] Miao He, Yanzhe Cui, Mohammad H. Mahoor, and Richard M. Voyles. A heterogeneous modules interconnection architecture for fpga-based partial dynamic re-

configuration. In *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–7, 2012.

[KMH+21] Dima Kilani, Baker Mohammad, Yasmin Halawani, Mohammed F. Tolba, and Hani Saleh. C3pu: Cross-coupling capacitor processing unit using analog-mixed signal in-memory computing for ai inference, 2021.

[KSJ22] O. Krestinskaya, K. Salama, and A.P. James. Analog image denoising with an adaptive memristive crossbar network. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3453–3457, 2022.

[LKS+18] Y. Li, S. Kim, X. Sun, P. Solomon, T. Gokmen, H. Tsai, S. Koswatta, Z. Ren, R. Mo, C. C. Yeh, W. Haensch, and E. Leobandung. Capacitor-based cross-point array for analog neural network with record symmetry and linearity. In *2018 IEEE Symposium on VLSI Technology*, pages 25–26, 2018.

[LSB21] Shih-Chii Liu, John Paul Strachan, and Arindam Basu. Prospects for analog circuits in deep networks. *CoRR*, abs/2106.12444, 2021.

[SGK17] Gerben M Scheepmaker, Rob MP Goverde, and Leo G Kroon. Review of energy-efficient train control and timetabling. *European Journal of Operational Research*, 257(2):355–376, 2017.

[TRBR19] A. Tadić, A. Reyes Borunda, and M. W. Ray. Development of a voltage controlled resistor for use in a self-balancing resistance bridge. *Review of Scientific Instruments*, 90(12), 12 2019. 124706.

[WATHES22] Shihao Wang, Karama M. Al-Tamimi, Issam Hammad, and Kamal El-Sankary. Towards current-mode analog implementation of deep neural network functions. In *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*, pages 322–326, 2022.

[Xue15] Yang Xue. Recent development in analog computation - a brief overview, 2015.

A. Partial Code

```
struct Shortcut{S}
    s::S
end
Flux.@functor Shortcut
(s::Shortcut)(mx, x) = mx + s.s(x)

struct ResidualBlock{B}
    block::B
end
Flux.@functor ResidualBlock
(b::ResidualBlock)(x) = relu.(b.block(x))

function BasicBlock(channels, connection; stride = 1)
```

```
    layer = Chain(
        Conv((3, 3), channels; stride, pad=1, bias=false),
        BatchNorm(channels[2], relu),
        Conv((3, 3), channels[2]=>channels[2]; pad=1, bias=false),
        BatchNorm(channels[2]))
    ResidualBlock(SkipConnection(layer, connection))
end

function Bottleneck(channels, connection; stride = 1, expansion = 4)
    layer = Chain(
        Conv((1, 1), channels, bias=false),
        BatchNorm(channels[2], relu),
        Conv((3, 3), channels[2]=>channels[2]; stride, pad=1, bias=false),
        BatchNorm(channels[2], relu),
        Conv((1, 1), channels[2]=>(channels[2] * expansion); bias=false),
        BatchNorm(channels[2] * expansion))
    ResidualBlock(SkipConnection(layer, connection))
end

function make_layer(block, channels, repeat, expansion, stride = 1)
    layer = ResidualBlock[]

    if stride == 1 && channels[1] == channels[2]
        push!(layer, block(channels, +; stride))
    else
        c = Shortcut(Chain(
            Conv((1, 1), channels; stride, bias=false),
            BatchNorm(channels[2])))
        push!(layer, block(channels, c; stride))
    end

    expanded_channels = channels[2] * expansion
    for _ in 2:repeat
        push!(layer, block(expanded_channels=>channels[2], +))
    end
    Chain(layer...)
end

struct ResidualNetwork{E, P, L, H}
    entry::E
    pooling::P
    layers::L
    head::H

    size::Int64
    stages::NTuple{5, Int64}
end
Flux.@functor ResidualNetwork

(m::ResidualNetwork)(x) = m.head(m.layers(m.pooling(m.entry(x))))
(m::ResidualNetwork)(x, ::Val{:stages}) = Flux.extraChain((
    m.entry, Chain(m.pooling, m.layers[1]),
```

```
    m.layers[2], m.layers[3], m.layers[4]), x)

function ResidualNetwork(in_channels = 3, classes = 1000)
    repeats = (2, 2, 2, 2)
    block = BasicBlock
    expansion = 1
    stages_channels = _get_stages_channels(expansion)

    entry = Chain(
        Conv((7, 7), in_channels=>64, pad=(3, 3), stride=(2, 2), bias=false),
        BatchNorm(64, relu))
    pooling = MaxPool((3, 3), pad=(1, 1), stride=(2, 2))

    head = nothing
    if classes    nothing
        head = Chain(MeanPool((7, 7)), Flux.flatten, Dense(512 * expansion, classes))
    end

    in_channels = 64
    channels = (64, 128, 256, 512)
    strides = (1, 2, 2, 2)

    layers = []
    for (out_channels, repeat, stride) in zip(channels, repeats, strides)
        push!(layers, make_layer(
            block, in_channels=>out_channels, repeat, expansion, stride))
        in_channels = out_channels * expansion
    end

    ResidualNetwork(
        entry, pooling, Chain(layers...), head,
        model_size, stages_channels)
end

@inline _get_stages_channels(expansion) = (
    64, 64 * expansion, 128 * expansion, 256 * expansion, 512 * expansion)
```