Evolutionary Exploration of an Ultrasound Propagation Predictor Neural Network *

Jakub Chlebik and Jiri Jaros

Faculty of Information Technology, Brno University of Technology,
Centre of Excellence IT4Innovations,
Bozetechova 2, 612 66 Brno,
Czech Republic
ichlebik@fit.vut.cz jarosjir@fit.vut.cz

Abstract. To find an optimal treatment plan for an focused ultrasound based procedure, a multitude of computationally expensive simulations need to be evaluated, often thousands of times. Recent renaissance of machine learning technologies could provide a solution to this problem, as a recently published article presented a Physics Informed Neural Net to predict Acoustic Propagation through a human skull. The learned optimizer showed an excellent performance on the test set, and is capable of generalization well outside the training examples, including to much larger computational domains, and more complex source and sound speed distributions. The utilized UNet architecture, however, was marked by the authors as the uncertain part of the design with a real possibility to improve upon. To explore their design more deeply and to confirm their theories, we made an attempt to improve the solver by use of an evolutionary algorithm, challenging the importance of different building blocks and authors original choices and decisions regarding the architecture. Two experiments using Cartesian Genetic Programming had been ran, first to try and optimize the original nets architecture, and a second one, to study the effects of the employed multi-resolution encoding on precision of the network. Our exploitative experiments managed to find a network with approximately an order of magnitude better RMSE for the predictor, using the same validation set as the original solver. The exploration evolution process then showed the use of 4 resolution layers as valid, and provided topics for further research on the effects of the memory blocks and convolution kernel sizes.

Keywords: Evolutionary Optimisation · Evolutionary Design · Ultrasound Propagation Predictor · Cartesian Genetic Programming.

* This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90140).This work was supported by the Czech Science Foundation project 21-13001S. This project has received funding from the European Unions Horizon Europe research and innovation programme under grant agreement No 101071008.

1 Introduction

A very promising alternative to the standard cancer treatment procedures is a non-invasive high intensity focused ultrasound (HIFU), also known as focused ultrasound surgery [13, 15, 3]. The technique works by sending a focused beam of ultrasound into the tissue and causing a reaction by energy exchange inside the focus. Indeed, in recent years, HIFU had been applied to treat a variety of solid malignant tumors in a well-defined volume, including the pancreas, liver, prostate, breast, uterine fibroids, and other soft-tissue sarcomas.

Furthermore, techniques of neurostimulation are also possible by using Low Intensity Focused Ultrasound (LIFU) instead of HIFU transducers. Neuromodulation mediated by LIFU has excellent advantages over conventional neuromodulation techniques such as deep brain stimulation, transcranial magnetic stimulation, and many others. It can be expected that all these conventional neuromodulatory techniques will be replaced in the future by the non-invasive LIFU method, with which we will be able to treat all the above-mentioned diseases [8, 1]. However, these methods have currently a lot of drawbacks, i.e. requiring an open surgery or having a low precision.

Using computational ultrasound models and knowledge of the properties of the medium, it is possible to predict the ultrasound field inside the tissue after propagating through it, and thus account for subject-specific dose and targeting variations [4]. However, existing models based on conventional numerical techniques typically take tens of minutes to several hours to complete due to the large size of the computational domain compared to the size of the acoustic wavelength, in some cases generating models with billions of unknowns which require tens of thousands of iterations to solve [9]. This makes them too slow to be used for online calculations and corrections, i.e., while the subject is undergoing the therapy.

Recent renaissance of machine learning technologies could provide a solution to this problem, as a recently published article [11] presented a Physics Informed Neural Net to predict Acoustic Propagation through human skull. While the utilized UNet is reasonably small, a multiple redundant parts are present within the design. Furthermore, authors themselves suggested more experimentation was needed with the architecture to explore the effects of different parts of the design.

To use this net in an ultrasound treatment plan optimization loop, precision and delivery speed is of the highest importance. In this spirit, we attempt to tune the architecture, optimizing the number of parameters while preserving or increasing the precision. With modern emergence of Neural Architecture Search (NAS) methods using evolutionary approaches managing to outperform hand-designed architectures [16, 7, 10], we employ cartesian genetic programming (or CGP in short) to explore and potentially improve the original solution.

2 Ultrasound Propagation Solver

For many therapeutic applications, the applied ultrasound signals are at a single frequency and last for many milliseconds or seconds, which is typically much longer than the time taken for the acoustic field to reach a steady-state. Thus, the wave propagation can be modeled by the heterogeneous Helmholtz equation subject to the Sommerfeld radiation condition at infinity (see Eq. (1)).

$$\begin{aligned} \left[\Delta^2 + \left(\frac{\omega}{c(r)}\right)\right] u(r) &= \rho(r) \\ s.t. \lim_{|r|\to\infty} |r|^{\frac{n-1}{2}} \left(\frac{\delta}{\delta|r|} - i\frac{\omega}{c_0}\right) u(r) &= 0. \end{aligned} \tag{1}$$

Here n is the number of spatial dimensions, c is the speed of sound, ω is the angular frequency of the source, r a general space coordinate. ρ is the source distribution and $u(r)$ is the complex acoustic wavefield.

$$\begin{aligned} (\Delta u^{k+1}, h^{k+1}) &= f_\theta(u^k, e^k, h^k) \\ u^{k+1} &= u^k + \Delta u^{k+1} \end{aligned} \tag{2}$$

where u denotes the wavefield we are solving for, h is a recurrent belief state, e_k represents the residual of the PDE being solved and f_θ is a learnable differentiable function approximating the Helmholtz equation.

2.1 Helmnet

The solution, we are attempting to improve, was presented by Stanziola et. al. in [11] and its aim is solving the 2D version of the Helmholtz equation. In this implementation, the boundary condition are satisfied by the use of a perfectly matched layer [2]. The training is guided using a physics-based loss, formed by the residual of the Helmholtz equation (see Eq. (1)) - this allows the solution to avoid labeled training data and, in turn, alleviates the need for supervised learning with large preexisting datasets. One distinctive feature of Helmnet is utilization of a replay buffer, which enables the model to be trained by unrolling for a large number of iterations.

Architecture Helmnet is designed as a recurrent UNet, describing the Eq. (2). Figure 1 shows the iterative solver pipeline, highlighting two iterations. Authors point out several intuitions behind their choice of the UNet architecture - having a fully convolutional network implicitly imposes some degree of translation invariance to the iterative solver, while at the same time allows the network to be used with arbitrarily-sized sound speed distributions. Furthermore, the network can encode priors at different scales thanks to the multiscale structure, allowing correction for very local distortions of the wavefield while at the same time taking care of long range dependencies.

The core building block of the network is a widely known double-convolution (DC) layer - two 2D convolutions with 3×3 kernels, interleaved by a PReLU activation function.

Each encoding block (EB) contains two DC layers (see Fig. 2). The first accepts two inputs: the network input representation at the current scale and a hidden state. The output is then passed to a second double-convolution layer used to update the hidden state, the corresponding decoding block of the network, and a restriction operator which downsamples the output and feeds it to a deeper resolution-layer of the network. The restriction operator is implemented by an 8×8 convolutional stage, applied with stride 2 in order to halve the dimension of the wavefield at each depth. Internally, each encoding block stores its own hidden state using a standard recurrent memory block.

The decoding blocks (DB) take an input from the layer below, upsamples it using transposed convolutions with 8×8 kernels and stride of 2, and after concatenating it with the output from the corresponding encoding block, produces an output via another double-convolution layer (see Fig. 2).

The last layer of the network is a 1×1 convolution that maps the output of the neural network to the wavefield domain. The output has the same spatial dimensions as the input and contains two channels for the real and imaginary parts of the wavefield.

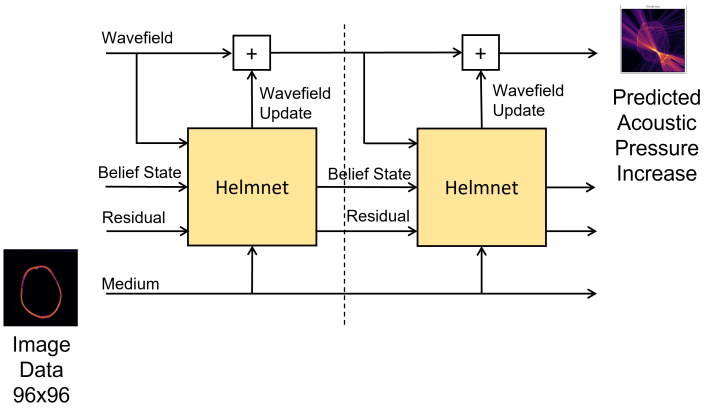


Fig. 1: Scheme of the iterative solver proposed by Stanziola et al. [11]. A belief state, a residual and the current-iteration wavefield, are passed as inputs to every iteration of the Helmnet.

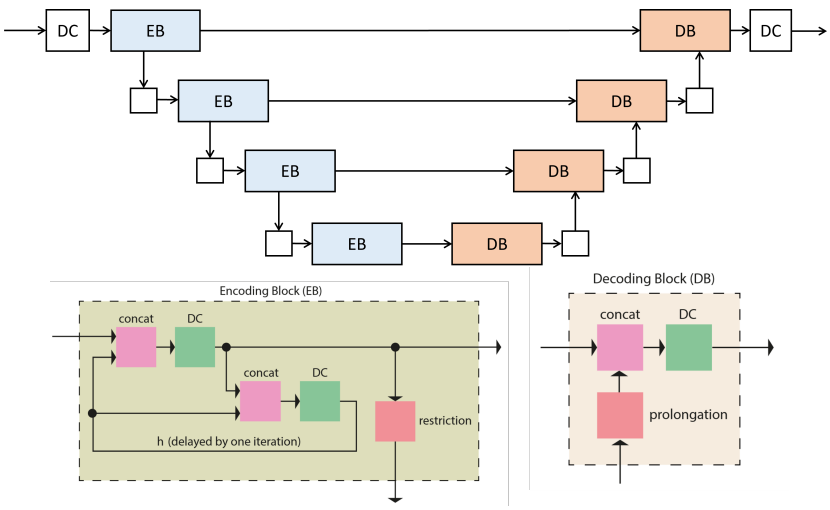


Fig. 2: A scheme of Helmnet - a 4 level deep UNet proposed by the authors of [11]. Four dimension layers, each encoding at different spatial dimensions. Every layer consists of an Encoding (EB) and a Decoding Block (DB), with a skip connection between them, as depicted bellow. Each encoding block contains two double convolution (DC) layers, one to compute the output passed to subsequent layers, and one to compute the hidden state h. Each decoding block consists of one DC block. The concatenation blocks in both sections stack the inputs on the channel dimension and the restriction/prolongation blocks perform a simple downscale/upscale for the layers bellow/above. The network is lightweight, with only 8 channels per convolution block at every scale and a total of $47k$ trainable parameters.

3 Experiments Setup

In this section, we will go over the base setup for our experiments, going over problem decomposition, encoding and parameters. We will be using Cartesian Genetic Programming [6] for the evolutionary exploration, as its form and properties lend itself very well for use with neural networks and general NAS uses [5, 14].

3.1 Decomposition

We aim to explore and potentially optimize these parts of Helmnet:

1. Challenge the use of full 4 layers of Encoder-Decoder design. Reducing the number of layers without meaningful loss of precision would greatly reduce the number of parameters of the network.
2. Challenge the use of a standard 3×3 Double Convolution layer with a memory block for Encoder and Decoder. A different architecture, such as Dense Layers, could yield better results.

3.2 Encoding

The entire net is encoded into a genotype as a sequence of integers. The first two genes encode the activation function and the downsampling operation for the entire network. After that follows a sequence representing a resolution layer of the UNet. Each sequence contains genes for the memory operation, encoder block operation, encoder block connections, decoder block operations and decoder block connections. This sequence is then repeated for every resolution layer.

Encoder and Decoder Blocks Each encoder block (EB) and decoder block (DB) is re-imagined for the purpose of CGP as a sequence of 4 arbitrary connected neurons, with EB also containing a memory module, see Figs. 5a, 5b. Each neuron inside EB and DB can be one of following operations:

- 3×3, 5×5 or 7×7 Convolution. All with option to skip an activation function.
- Linear layer, to allow for intermediary result scaling.
- Identity operation with no trainable parameters, effectively disabling the neuron.

See Fig. 4a for a visualisation of this set.

Memory Module Memory module inside the encoder block contains a single operation and can be one of the following:

- 3×3, 5×5 or 7×7 Double Convolution. The DC layer, instead of a simple CONV layer, was chosen to allow for a finer control of the memory encoding process.
- Disable the memory module. A memory is required for the recurrent network to function, however, UNet is specific for encoding data at multiple levels of resolution. Each layer in the original net contained a memory module. The effect of encoding previous states for each resolution is one of the topics of our research and thus we allow the search to disable memory at different levels. While this makes it theoretically possible to disable memory entirely, such an individual would not have a good fitness and should be eliminated by selection. This option is present inside the set twice (see Fig. 4b) to nudge the evolution towards candidates with some modules disabled.

Downscaling Module Downscaling module is selected globally and will remain the same for the entire net. E.g. if *Average Pooling* is chosen as the downscaling option, every downscaling module will perform average pooling, with their own separate set of parameters. Figure 4c pictures the available options for downscaling.

Activation One activation function is also selected globally for the entire network. Multiple different activation functions can have an effect on the performance of the network, however, allowing multiple activation functions inside the network increases the already big search space substantially, without any provable benefit, while also not being the topic of our research. Activation can be seen on Fig. 4d.

Operation Connections Each Encoding and Decoding block consists of 4 neurons, each executing their own function from the predetermined set. Connections between these, and thus their inputs and outputs, are also subjected to the evolution process. The connections are encoded as binary strings, with each bit representing an input from the previous neuron, starting with the input to the block itself. In this way, we can create a linear progression throughout the layer, as well as a skip connections known from ResNets. An example of a connection string can be seen in Figure 5. Additionally, rules were introduced to solve cases of half-orphaned neurons (nodes with inputs but no outputs or vice-versa):

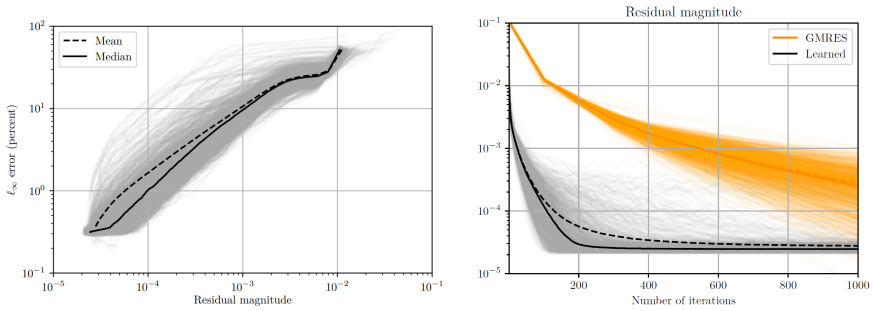
- Nodes with multiple inputs have them summed together before executing.
- A node with an output connection but no input connection gets the module input as its input.
- Output of the module is a sum of all nodes that were activated but had no outgoing connections.

3.3 Fitness Function and Training Setup

With the encoding outlined, next step was to create a fitness function. Since we are trying to improve an already existing neural net, each individual needs to be at least partially trained and evaluated using the same dataset as the original Helmnet [12]. This dataset consists of 10000 samples of 96×96 sound speed distributions of idealized, artificially generated, skulls. The training-validation split was kept the same, at 9 to 1. Helmnet was trained for a 1000 epochs, using a 1000 iterations to reach a steady-state for each sample. Sadly, we cannot afford to fully train each candidate solution as the hardware resources and time required to train each candidate would take far too long. As such, we decided on a few compromises:

- We are using 200 iterations per sample, instead of the original 1000. Results of the original publication [12] show that the net reach close to steady-state solution somewhere between 200 and 300 iterations, with the following iterations only marginally contributing to decrease in error. See Fig.3b.
- We are training each candidate for just 20 epochs. While, it is true that training for more epochs provides more reliable results, 20 epochs should be good enough to provide a baseline idea about the candidates performance.

After training the candidate, its residual on the validation set forms the candidate's fitness. Stanziola showed that residual is directly correlated to predictor precision, see Fig. 3a. Other training and network parameters, such as the number of channels, the momentum, the batch size and so on, were preserver.



(a) (b)

Fig. 3: Two of many graphs lifted from the original work of Stanziola et al. [11]. (a) is a graph showing how the residual of the Helmholtz solver is directly correlated to the prediction error. (b) was originally meant to show the difference between GMRES and their trained solver. However, this figure also shows that roughly only 200 to 300 iterations were required to get close to the final solver error.

3.4 Evolution Parameters

Crowds To separate solutions based on the enabled and disabled resolution layers, the solutions are separated into crowds. The candidates with the same resolution 'rows' enabled are considered to be a part of the same crowd.

Self-Adaptation Experiment To improve the original architecture, a Self-Adaptation experiment was performed. The main purpose of this experiment was exploitation. Here, the evolution process itself was given a chance to change the crowd of one solution to a different one. If an individual is selected for this process, its crowd is randomly mutated, thereby making some of its layers a simple pass-through layers, effectively disabling them. The main/primary resolution is an exception to this and will never be disabled. This is done to satisfy the sampling theorem based on the frequency of the simulated acoustic waves source and the necessary calculation of the Laplacian to solve the Helmholtz equation (see Eq. 1). Disabled layers are not considered for standard mutation operations.

The evolution parameters were chosen as follows:

- 20 generations for the evolution process.
- As the main purpose of this experiment was to optimize the current architecture and promote exploitation, a standard $(1 + \lambda)$ scheme, with $\lambda = 15$, was used.
- Offspring are generated by mutation, which is implemented as random change of a selected gene to a different value. Two mutation rates are used:
  - One for a standard mutation - 15% of the parent solution is changed to create an offspring. While 15% is higher than the usually chosen values,

we feel its justified based on the length of the chromosome and the need to introduce a sufficiently large change. Many genes are not independent on each other, i.e. blocks and the connection string have a close tie with each other, and a small change might not meaningfully impact the performance. Changing 15% of genes corresponds to a full swap of one of the 8 UNet blocks. Genes to mutate are picked randomly with a uniform distribution; exception being any disabled layers. Genes inside a the disabled layer have a 0% chance of being picked.

- The second one to mutate the crowd of the parent - 10% of the offspring will have their crowd changed and thus some of their resolution layers disabled. The main/primary resolution is an exception to this and will never be disabled.

Co-Evolution Experiment To investigate the architecture of the solver, a Co-Evolution experiment was run. This experiment's main purpose is to explore different crowds, as such we starts with an overall much bigger population ($\lambda = 48$) and a uniform spread of individuals, for each crowd across the population. With each generation, a step of Simulated Annealing is executed on the ratio of the population space occupied by each crowd, taking some members of a badly-performing crowd and assigning their spots inside the population to a better performing one. Performance of a crowd is measured as a median fitness of individuals inside that crowd. Since the crowds can get pretty small, historical data, going one generation back, are also considered.

The evolution parameters were chosen as follows:

- 20 generations for the evolution process.
- Main purpose for this experiment is an exploration of different crowds, as such, an $(8, \lambda)$ scheme, with $\lambda = 64$, was used. Here, each crowd has its own parent (so 8 parents in total). The ration of split of lambda is starting at $\frac{\lambda}{8}$ for each and is consequently optimized by a cooperating Simulated Annealing algorithm, with a linear cooling scheme and $T_{start} = 100$. We decided on SA because of its overall known solid performance and the ability to pick a worse performer to promote exploration, which can be very important for the first few generations. Performance of a crowd split distribution is measured as a weighted sum of performances of each crowd.
- Tournament selection of size 2 was used to pick crowds competing for a spot inside the population and generate a candidate crowd split distribution.
- A standard mutation - 15% of the parent solution is changed to create an offspring. Genes to mutate are picked randomly with a uniform distribution; exception being any disabled layers. Genes inside the disabled layer have no chance of being picked. Mutation is implemented as a random change to a different value.

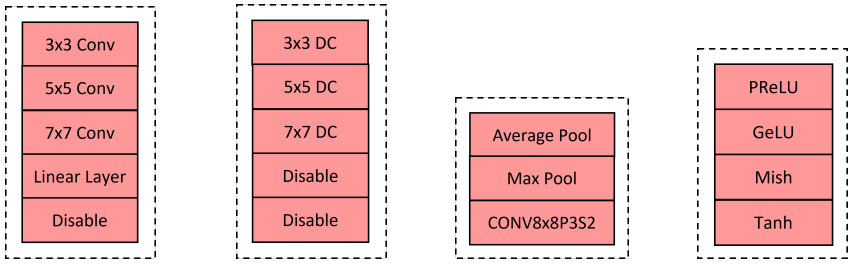


Fig. 4: Sets of operations allowed for different blocks inside the architecture - (a) EB and DB operations. Each Conv operation additionally may or may not be followed by an activation. (b) Memory operations (c) Downscaling operations and (d) Activation functions. Both (c) and (d) are a global choice, meaning only one variant of the activation and downscaling is used for the entire net.

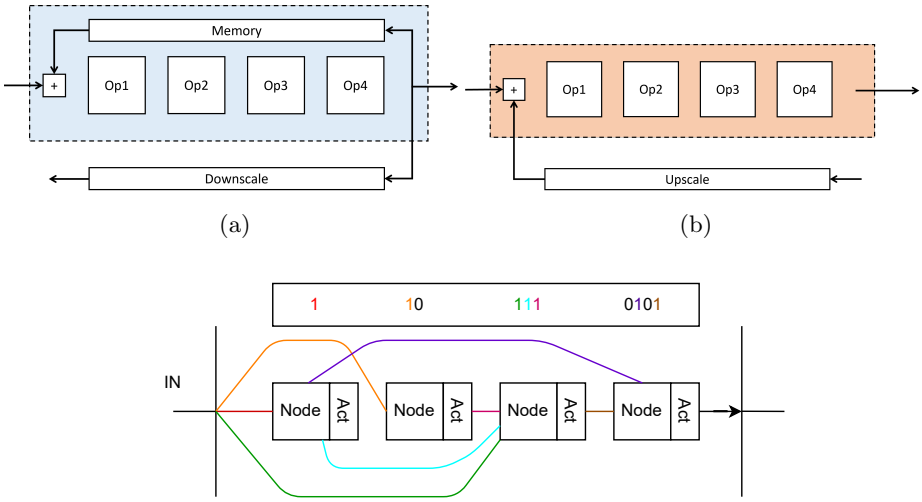


Fig. 5: Cartesian Genetic Programming phenotypes representing the (a) encoder block with memory (EB) and the (b) decoder block (DB). Connections between input, evolved operations and output are represented as a binary combination string and are a part of the evolution process for both blocks. An example of a connection can be seen on the bottom figure.

4 Results

With the setup of both experiments outlined, this section presents the results. Original HelmNet solution was encoded and used as a seed solution for the first

generations of our experiments. Both experiments were repeated 5 times, and most figures presented are showing the aggregate results of those runs.

Each candidate net was trained for 20 epochs on the entire training set outlined previously, performing 250 iteration per sample, before proclaiming the solution reached steady-state. The performance on the validation set was then considered as the fitness value of the candidate.

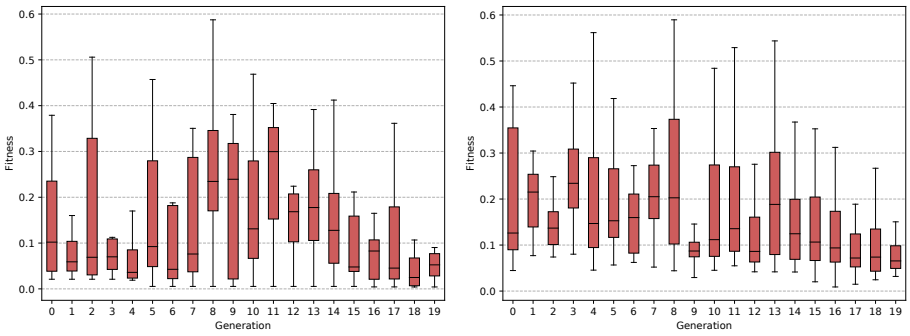
Technologies The individuals were trained using PyTorch 1.9 and close to state-of-the-art hardware:

- 8× NVIDIA A100 40 GB GPUs
- 1024 GB RAM
- 2× AMD Zen 3 EPYC™ 7763

Using this hardware, training and evaluation of a single candidate solution took roughly 5 minutes, with about 24 hours per one run of an experiment.

4.1 Evolution Run

Figures 6a and 6b show the progression of the evolution process for both of our experiments. As expected, given its focus on exploitation, the Self-Adaptation setup managed to find better solutions. Exploring the different combinations of resolution-encoding layers, the figure 8a shows the boxplot fitness values for each crowd separately, aggregated from the last generations of all Co-Evolution runs. Crowd 7, the original resolution-encoding architecture, seems to be standing above other, but crowds 0 - all but the main layer disabled and crowd 4 - only the two upper layers could be explored more in future, as their results are not that far away.

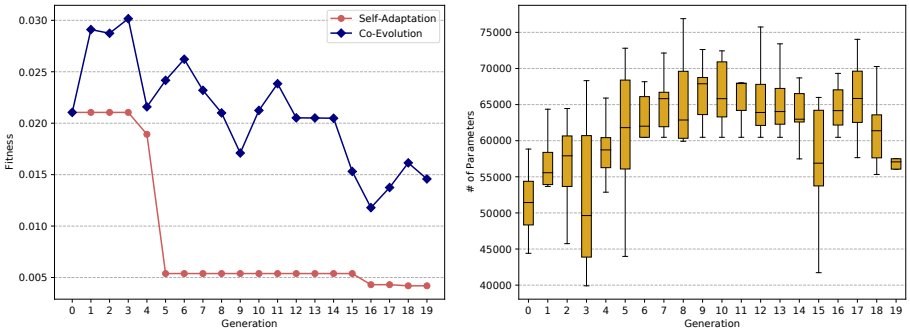


(a) (b)

Fig. 6: Boxplots of the evolution runs from the (a) Self-Adaptation experiments, and the (b) Co-Evolution experiments.

4.2 Best Solution

Figure 7a show the progression of the best individuals found by each experiment setup. As expected, Self-Adaptation experiment proved to be better at improving the original Helmnet solution. After training this individual fully, the resulting RMSE error is one order of magnitude better than the original solution (10^{-4} vs 10^{-5}). The figures also show the effect of different evolution schemes - the plus scheme introduced elitism in an attempt to leverage exploitation. This approach kept the best solution performance constant throughout multiple generations, compared to the comma scheme, where we allowed for even a worse solution to become a parent but promoting exploration. Curiously, when using a crowd scheme different than 0, the biggest jump in fitness occurred with the removal of a memory block at the main resolution layer. All the best solutions from every experiment show this trait. Disabling this block corresponds with the big fitness drops in fig. 7a. For the Self-Adaptation experiment between generations 4 and 5, and between generations 14 to 15 in the Co-Evolution experiment. We believe this pattern is worth investigating in the future. Figure 7b shows the parameters spread of the best individuals taken from the exploitative Self-Adaptation runs. We can see that evolution tends to increase the number parameters to produce individuals with better fitness.

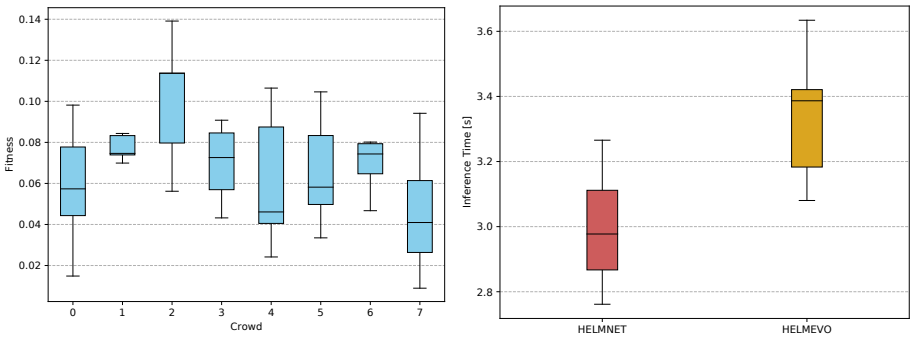


(a) (b)

Fig. 7: Plots showing (a) the progression of the best individuals found by the Self-Adaptation and the Co-Evolution experiments and (b) the parameters progression of the best individuals during the best Self-Adaptation experiment.

4.3 Used Blocks

Figure 9 shows histograms of used function blocks inside each of the encoding and decoding blocks, shown for each resolution layer separately, taken from the last generations of the SA experiments. In general, we can roughly see the preference of high resolution encoding blocks to use the larger convolution kernels,



(a) (b)

Fig. 8: Left figure (a) is showing fitness values of individual crowds, taken as an aggregate from the last generations of the Co-Evolution experiments. Figure (b) is comparing inference times of the original Helmnet and our evolution optimized version Helmevo.

decreasing with the resolution on lower layers. Evolution tent to not use the full capacity of the blocks, a lot of the functions are being changed into identities on different layers. Additionally, it appears the evolution is doing the main encoding and decoding on different layers, creating some structures commonly found in ResNets. Furthermore, linear layers are very rarely used, no matter the layer.

For the main resolution layer, a big part of decoding blocks was disabled, with the rest equally using the smallest and the biggest convolution kernels.

For the second resolution layer, almost all of the encoder block is disabled, only leaving a few Conv blocks, mostly still C7s. As for the decoder blocks, we see the biggest use of linear layers, while continuing the same trend from the prior resolution - C3s and C7s at similar distributions.

The results from the the third layer are less clear, however, we can still try to draw some conclusions. The 24×24 layer prefers smaller kernels, probably as the result of smaller resolution - C7 would take almost a third of the entire domain. Note, that decoder blocks still greatly prefer C5s and C7s.

The smallest resolution is again very clear - a lot of disabled blocks for both the EB and DB, with decoder preferring smaller kernels and encoder preferring either C3 or C7. Note that unlike other decoders, the lowest DB does not have to deal with two inputs.

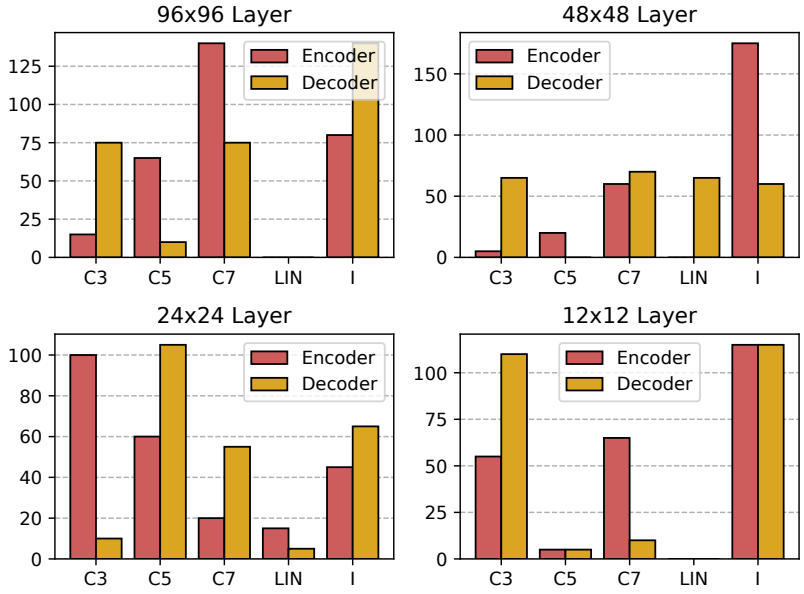


Fig. 9: Histograms of blocks used inside the encoding and decoding stages during the last generations, separated by resolution layers of the UNet. Cx stands for convolution with the kernel size of x, LIN is a linear scaling block and I is an identity block. Identity blocks are never followed by an activation, and should therefore be considered as disabled.

5 Conclusion

5.1 Evolutionary Optimization

The Self-Adaptation experiment managed to find a better solution than the original - when we took this candidate and trained it fully for $1,000$ epochs using $1,000$ iterations, just as the original solution was, the resulting net showed approximately 1 order of magnitude better RMSE (10^{-4} vs 10^{-5}) than the original UNet. However, our solution is using $12,000$ more parameters. Figure 8b shows the comparison between our evolution optimized solution and the original Helmnet time to solution. It is clear that the increase in precision comes at the cost of increased inference time. Considering the speed was not a part of the fitness evaluation, it is to be expected. Undeniably, Helmnet is faster, however, if the requirements for the focused ultrasound planning calls for real-time live updates and navigation, neither of the solutions are fast enough. On the other hand, if we can afford some planning time in advance, the difference in inference time will not be meaningful enough. Admittedly, both solutions will need to transition into a 3D domain before we can make a final decision.

5.2 Crowding and Architecture Exploration

While the Co-Evolution experiment did not manage to improve upon the existing solution, the crowding approach allowed us to take a look at how disabling different resolution layers influence the quality of a generated individual. Measurably, crowd 7 - the original design of UNet, with all resolution layers enabled - generated the best candidate solutions. As suspected, this crowd is understandably using the most parameters for its candidates on average, indicating once again that increase in parameters seem to be a good way forward if we wish increase the networks accuracy further.

Furthermore, figure 9 shows a very interesting spread of convolution window sizes, with the encoder blocks preferring different kernel sizes than the decoders. Additionally, the search algorithm disabled almost entirety of some encoding or decoding blocks at different resolution layers, thus creating structures usually found in ResNets.

5.3 Memory Blocks and Future Research

Furthermore, while not the main focus of our experiments if we take a more detailed look at the memory blocks inside the candidate solutions, we find two things:

- The best solution found by all types of experiments and tries have the memory block at the main resolution layer disabled.
- The removal of said block from the main layer corresponds to a drop in residual of the candidate and increase in fitness in most cases throughout the evolution process.

Admittedly, the sample size is too small to make a resolute statement about this issue. The observation might, however, serve as a good topic for future research.

References

1. Abrahao, A., Meng, Y., Llinas, M., Huang, Y., Hamani, C., Mainprize, T., Aubert, I., Heyn, C., Black, S.E., Hynynen, K., Lipsman, N., Zinman, L.: First-in-human trial of blood-brain barrier opening in amyotrophic lateral sclerosis using MR-guided focused ultrasound. Nat. Commun. **10**(1), 4373 (Sep 2019)
2. Berenger, J.P.: A perfectly matched layer for the absorption of electromagnetic waves. Journal of Computational Physics **114**(2), 185 – 200 (1994). https://doi.org/https://doi.org/10.1006/jcph.1994.1159, http://www.sciencedirect.com/science/article/pii/S0021999184711594
3. Cudova, M., Treeby, B.E., Jaros, J.: Design of hifu treatment plans using an evolutionary strategy. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. p. 1568–1575. GECCO '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3205651.3208268, https://doi.org/10.1145/3205651.3208268

4. Lee, W., Kim, H.C., Jung, Y., Chung, Y.A., Song, I.U., Lee, J.H., Yoo, S.S.: Transcranial focused ultrasound stimulation of human primary visual cortex. Sci. Rep. **6**(1), 34026 (Sep 2016)
5. Miller, J.F.: Cartesian Genetic Programming, pp. 17–34. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17310-3_2, https://doi.org/10.1007/978-3-642-17310-3_2
6. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) Genetic Programming. pp. 121–132. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
7. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. CoRR **abs/1802.01548** (2018), http://arxiv.org/abs/1802.01548
8. Rezayat, E., Toostani, I.: Review paper: A review on brain stimulation using low intensity focused ultrasound. Basic and Clinical Neuroscience Journal **7** (07 2016). https://doi.org/10.15412/J.BCN.03070303
9. Robertson, J.L.B., Cox, B.T., Jaroš, J., Treeby, B.E.: Accurate simulation of transcranial ultrasound propagation for ultrasonic neuromodulation and stimulation. The Journal of the Acoustical Society of America **141**(3), 1726–1738 (mar 2017). https://doi.org/10.1121/1.4976339
10. Stanley, K., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence **1** (01 2019). https://doi.org/10.1038/s42256-018-0006-z
11. Stanziola, A., Arridge, S.R., Cox, B.T., Treeby, B.E.: A helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound. Journal of Computational Physics **441**, 110430 (sep 2021). https://doi.org/10.1016/j.jcp.2021.110430, https://doi.org/10.1016%2Fj.jcp.2021.110430
12. Stanziola, A., Arridge, S.R., Cox, B.T., Treeby, B.E.: A helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound. Journal of Computational Physics p. 110430 (2021). https://doi.org/https://doi.org/10.1016/j.jcp.2021.110430, https://www.sciencedirect.com/science/article/pii/S0021999121003259
13. Suomi, V., Jaros, J., et al.: Full modeling of high-intensity focused ultrasound and thermal heating in the kidney using realistic patient models. IEEE Transactions on Biomedical Engineering **PP**, 1–1 (09 2018). https://doi.org/10.1109/TBME.2018.2870064
14. Vasicek, Z., Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genetic Programming and Evolvable Machines **12**(3), 305–327 (Sep 2011). https://doi.org/10.1007/s10710-011-9132-7, https://doi.org/10.1007/s10710-011-9132-7
15. Zhou, Y.: High intensity focused ultrasound in clinical tumor ablation. World journal of clinical oncology **2**, 8–27 (01 2011). https://doi.org/10.5306/wjco.v2.i1.8
16. Ünal, H.T., Basciftci, F.: Evolutionary design of neural network architectures: a review of three decades of research. Artificial Intelligence Review (07 2021). https://doi.org/10.1007/s10462-021-10049-5