An End-to-End Embedded Neural Architecture Search and Model Compression Framework for Healthcare Applications and Use-Cases

Bharath Srinivas Prabakaran and Muhammad Shafique

1 Introduction

As discussed in chapter "Massively Parallel Neural Processing Array (MPNA): A CNN Accelerator for Embedded Systems", deep learning has revolutionized domains worldwide by improving machine understanding and has been used to develop state-of-the-art techniques in fields like computer vision [15], speech recognition and natural language processing [21], healthcare [9], medicine [49], bioinformatics [20], etc. These developments are primarily driven by the rising computational capabilities of modern processing platforms and the availability of massive new annotated datasets that enable the model to learn the necessary information. Fields like medicine and healthcare generate massive amounts of data, in the order of hundreds of exabytes is the USA alone, which can be leveraged by deep learning technologies to significantly improve a user's quality of life and obtain substantial benefits. Furthermore, healthcare is one of the largest revenue-generating industries in the world, requiring contributions upward of 10% of the country's Gross Domestic Product (GDP) annually [4]. Countries like the United States routinely spend up to 17.8% of their GDP on healthcare [35]. The global health industry is expected to generate over $10 trillion revenue, annually by 2022, which is a highly conservative estimate as it does not consider the increasing global elderly population percentages [47]. The rising global average life expectancy is another byproduct of the substantial technological advancements in medicine and healthcare [34]. The Internet of Things (IoT) phenomenon serves as an ideal

B. S. Prabakaran (✉)
Institute of Computer Engineering, Technische Universität Wien (TU Wien), Vienna, Austria
e-mail: bharath.prabakaran@tuwien.ac.at

M. Shafique
Engineering Division, New York University Abu Dhabi, Abu Dhabi, UAE
e-mail: muhammad.shafique@nyu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_2

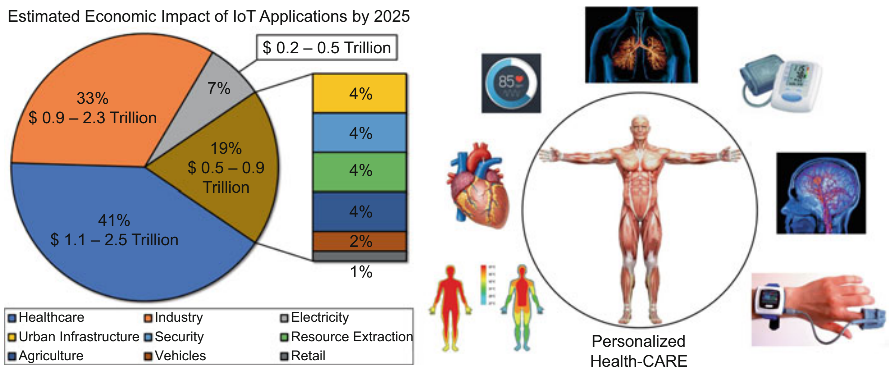

Fig. 1 Breakdown of the estimated economic impact of Internet of Things applications by 2025; an overview of the human bio-signals that can be monitored and analyzed for patient-specific care

opportunity that can be exploited by investigating its applicability in the healthcare sector to offer more efficient and user-friendly services that can be used to improve quality of life. The Internet of Medical Things (IoMT) market is expected to grow exponentially to achieve an annual economic impact of \$1.1–\$2.5 trillion by 2025, which constitutes 41% of the impact of the complete IoT sector [29]. This includes applications like personalized health monitoring, disease diagnostics, patient care, and physiological signal, or bio-signal, monitoring, and analytics to recommend person-specific lifestyle changes or health recommendations [2, 19], especially by investing heavily in the capabilities and advancements of deep learning. A breakdown of the estimated economic impact of IoT applications by the year 2025 and an overview of human bio-signals, which can be monitored and analyzed, are presented in Fig. 1.

An overview of a few healthcare use-cases and applications is discussed next, before moving on to the framework that can be used for exploring the deep learning models that can be deployed for a given use-case, given its output quality requirements and hardware constraints of the target execution platform.

1.1 Deep Learning in Healthcare: Potential Use-Cases and Applications

Medical Imaging Deep learning has been largely investigated as a solution to address research challenges in the computer vision and imaging domains due to the availability of massive labeled and annotated datasets. Therefore, the primary healthcare domain suitable for investigating the applicability of deep learning would be medical imaging. Since various technologies like X-Rays, CT (Computed Tomography) and MRI (Magnetic Resonance Imaging) scans, ultrasound, etc. are regularly used by clinicians and doctors to help patients, deep learning can be highly beneficial in such scenarios where they can be deployed as clinical assistants that can aid in diagnostics and radiology.

Electronic Health Record Analysis Electronic Health Record (EHR) is a collection of health data related to a patient across time, including their medical history, current and past medications, allergies, immunization information, lab test results, radiology imaging studies, age, weight, etc. These EHRs, from several patients, are combined into a large pool, based on their demographics, to mine and extract relevant information that can be used to devise new treatment strategies and improve the health status of the patients. Deep learning techniques have successfully demonstrated the ability to combine all this information to extract vital health statistics, including the ability to predict a patient's mortality.

Drug Discovery The processing capabilities of deep learning models can also be leveraged on massive genomic, clinical, and population-level data to identify potential drugs or compounds that can, "by-design," explore associations with existing cell signaling pathways, pharmaceutical and environmental interactions, to identify cures for known health problems. For instance, the protein folding problem, which had fazed the community for more than five decades, was recently solved by the AlphaFold deep learning model, proposed by researchers from Google [20]. This enables researchers to predict the structure of a protein complex, at atomic granularity, using just its amino acid composition, which can further enable scientists to identify compounds that can prevent the formation of lethal proteins in hereditary medical conditions like Alzheimer's or Parkinson's.

Precision Medicine Genomic analysis, in combination with drug discovery approaches, might be the key to develop the next generation of precise targeted medical treatments, which improve the user's quality of life. Understanding the genetic capability of the underlying condition, such as the type of cancer, its ability to reproduce, and the way it propagates, can enable scientists to better develop user-specific treatment options. However, processing such large amounts of data can take anywhere from weeks to months, which can be circumvent by deep learning models to the order of hours, enabling such explorations.

Similarly, there are plenty of other healthcare applications, like real-time monitoring and processing of bio-signals, sleep apnea detection, detecting gait patterns, genomic analysis, artificial intelligence-based chatbots and health assistants, and many more, that can benefit by investigating the applicability of deep learning in these use-cases (Table 1). We delve into the field of deep learning for healthcare,

Table 1 A summary of key state-of-the-art techniques in deep learning for healthcare

Deep learning in healthcare	References
Medical imaging	[1, 10, 15, 25, 27, 43]
Electronic health record analysis	[18, 39, 45, 46, 50]
Drug discovery and precision medicine	[7, 20, 24, 36, 38, 40, 42]
Others	[17, 23, 37]

next, by presenting a comprehensive embedded neural architecture search and model compression framework for healthcare applications and illustrate the benefits by evaluating its efficacy on a bio-signal processing use-case.

2 Embedded Neural Architecture Search and Model Compression Framework for Healthcare Applications

Figure 2 illustrates an overview of the deep neural network (DNN) model search and compression framework for healthcare, which is composed of six key stages. The framework considers (1) the user specifications and quality requirements, such as the required prediction labels, output classes, expected accuracy or precision of the model and (2) the hardware constraints of the target execution platform, such as the available on-chip memory (MBs) and the maximum number of floating-point operations (FLOPs) that can be executed per second to construct the required dataset from the existing labeled annotations (more details provided in Sect. 2.3) and explore the design space of DNN models that can be useful for the application.

2.1 User Specifications and Requirements

The framework enables dynamic model exploration by restricting the output classes of the DNN, based on the user requirements; besides the normal and anomalous classes, the user might require an output class specific to the target use-case. For instance, a hospital might require the model to classify X-Rays or CT scans to explicitly detect cases of lung infection caused by the novel SARS-CoV-2 coronavirus as a separate classification. These instances can be included by the framework to generate and explore DNN models specific to this case, given that the corresponding annotated data is already present in the dataset used for construction.

The framework currently considers four metrics for evaluating the quality of a model (Q), namely, Accuracy (A), Precision (P), Recall (R), and F1-score (F). Accuracy of a model is defined as the ratio of correct classifications with respect to the total number of classifications, Precision signifies the percentage of classified items that are relevant, Recall is defined as the percentage of relevant items that are classified correctly, and F1-score is used to evaluate the Recall and Precision of a model by estimating their harmonic mean. The system designer can specify a constraint in the framework using any of the abovementioned metrics, while exploring the DNN models for the application to ensure that the models obtained after exploration satisfy the required quality constraint. Evaluation with other use-case specific metrics is orthogonal to these standards and can be easily incorporated into the framework. These metrics are estimated as follows:

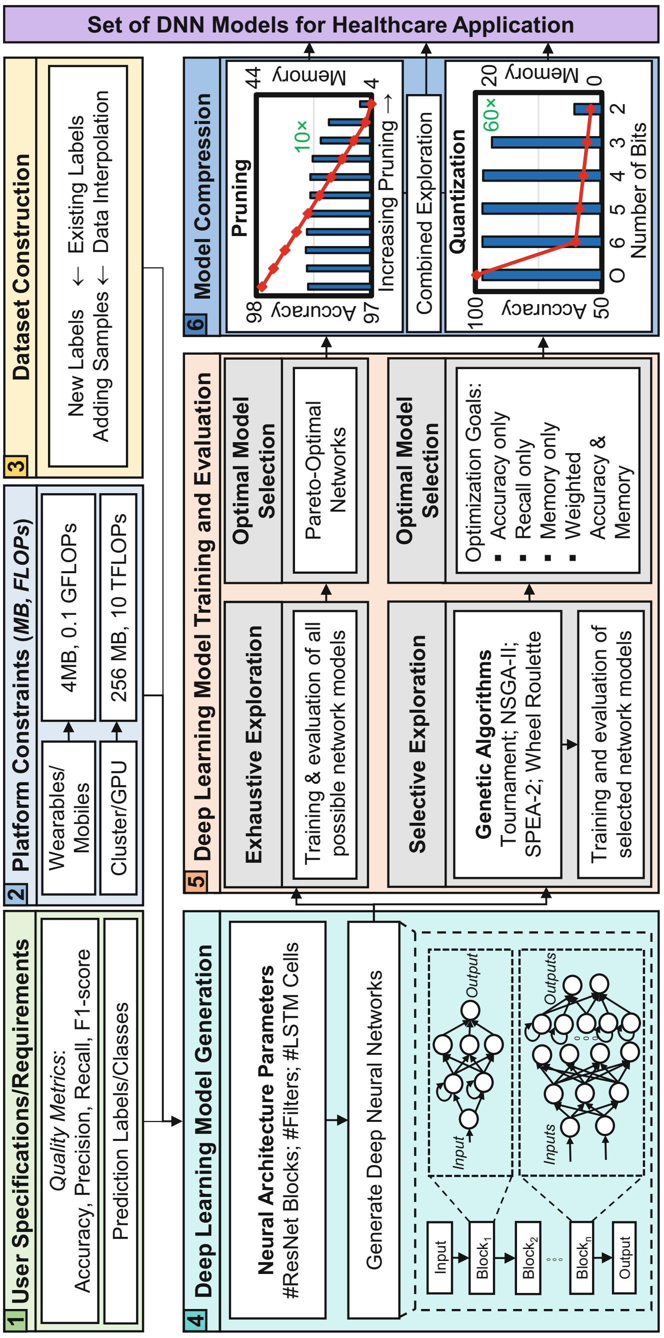
Set of DNN Models for Healthcare Application
1 User Specifications/Requirements
Quality Metrics: Accuracy, Precision, Recall, F1-score
Prediction Labels/Classes
2 Platform Constraints (MB, FLOPs)
Wearables/ Mobiles
4MB, 0.1 GFLOPs
Cluster/GPU
256 MB, 10 TFLOPs
3 Dataset Construction
New Labels ← Existing Labels
Adding Samples ← Data Interpolation
4 Deep Learning Model Generation
Neural Architecture Parameters
#ResNet Blocks; #Filters; #LSTM Cells
Generate Deep Neural Networks
Input
Block1
Block2
Blockn
Output
Input
Output
Inputs
Outputs
5 Deep Learning Model Training and Evaluation
Exhaustive Exploration
Training & evaluation of all possible network models
Optimal Model Selection
Pareto-Optimal Networks
Selective Exploration
Genetic Algorithms
Tournament; NSGA-II; SPEA-2; Wheel Roulette
Training and evaluation of selected network models
Optimal Model Selection
Optimization Goals:
Accuracy only
Recall only
Memory only
Weighted Accuracy & Memory
6 Model Compression
Pruning
98
97
Accuracy
44
4
Memory
10×
Increasing Pruning →
Combined Exploration
Quantization
100
50
Accuracy
20
0
Memory
60×
0 6 5 4 3 2
Number of Bits

Fig. 2 An overview of the key stages in the deep learning model exploration and compression framework for healthcare applications (adapted from [37])

$$A = \frac{TP + TN}{\#Classifications}, P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, F = \frac{2 * P * R}{P + R}$$

where TP stands for the number of true positive predictions, whereas TN, FP, and FN depict the number of true negative, the number of false positive, and the number of false negative predictions, respectively.

2.2 Platform Constraints

Similarly, to ensure that the explored networks do not require computational resources beyond those available on the target execution platform, a hardware constraint is implemented before the evaluation stage, which the user can define as per their requirements. Currently, the hardware constraints of the system can be specified by the system designer either in terms of the memory overhead (B), i.e., the maximum size of the model that can be accommodated on the platform, or the execution time in terms of the maximum number of floating-point operations that the platform can execute for a single inference (FP). Like the quality metrics, incorporating other additional platform-specific hardware constraints are orthogonal to our current approach and can be an easily added functionality to the framework. Explicitly specifying hardware constraints requires the framework to identify models that offer the best quality under the given constraints, which enables the exploration of a trade-off between output quality and hardware requirements of the model, two metrics that typically maintain an inverse correlation.

2.3 Dataset Construction

To ensure that the model developed is application-driven, a custom dataset is constructed by fusing labels, in an existing healthcare dataset, in order to create the required output classes. Note that each label in the custom dataset needs to be correspond to one of the labels in the existing healthcare dataset, to ensure coherence. For instance, with respect to the COVID-19 classifier application discussed earlier, there could be varying diagnosis for the lung X-Rays or CT scans present in the dataset, including pneumonia, pleural effusion, cystic fibrosis, or lung cancer, which are ultimately labeled as “anomaly” in the constructed dataset, given the sole focus of the application is to just classify amongst normal, anomaly, and “COVID-19.” A similar methodology can be used to construct custom datasets for a given healthcare application, as discussed with the help of a use-case in Sect. 3.

2.4 Deep Learning Model Generation

With the necessary information regarding the user specifications, platform constraints, and the constructed dataset, we generate the set of possible DNN models (ψ) by varying the key neural architecture parameters. Since our approach considers a relevant state-of-the-art model as a baseline, we extract the key architectural parameters from the baseline model and vary them to generate different models that can achieve (near) state-of-the-art accuracy with reduced hardware requirements. For instance, the use-case discussed in Sect. 3 explores three DNN model parameters, namely, (1) No. of Residual Network Blocks (#ResNet Blocks), (2) No. of Filters (#Filters), and (3) No. of LSTM Cells (#LSTM Cells), which can, theoretically, be any value in the $\mathbb{R}^+$ domain, leading to an explosion of the designs that need to be explored under an unbounded design space. By considering the state-of-the-art model as the upper bound, we restrict the number of designs to be explored, thereby ensuring that the algorithm converges in finite time. Furthermore, since the exploration of the models is heavily dependent on the state of the art, any modifications to the block-level structure of the baseline model, including changes to the *block*, will affect the design space of the models (ψ) to be explored.

2.5 Deep Learning Model Training and Evaluation

The DNN models generated earlier need to be trained and evaluated on the constructed dataset, individually, before their real-world deployment. However, since the training and evaluation of each individual DNN in the design space is a compute-intensive and time-consuming task, first, we need to reduce the number of models generated, which we ensure by constraining the hardware requirements of the model (as discussed earlier in Sect. 2.2), and second, we need to quickly explore the design space of DNNs, to reduce the overall duration of the task. Exploration of the design space, in our framework, can be conducted either exhaustively or selectively, as discussed below:

(1) Exhaustive Exploration It requires each individual model of the design space to be trained and evaluated on the constructed dataset in order to determine the set of Pareto-optimal DNN models, which essentially trade off between output quality and hardware requirements. The hardware constraints imposed by the execution platform combined with the state-of-the-art imposed upper bound enable the framework to exhaustively explore the design space of DNN models in tens of GPU hours, as opposed to hundreds or thousands of hours in the case of unconstrained exploration. Therefore, when the complexity of the model, its parameter format, the number of weights and biases, and the variation in the hyper-parameters increase, it is recommended to selectively explore the design space to circumvent the exponential rise in design space models. Exhaustive exploration is primarily included as a functionality to illustrate the efficacy of the selective exploration technique that has been discussed next.

(2) Selective Exploration It involves the effective selection, training, and evaluation of a small subset of the models in ψ in order to reduce the exploration time to a couple of GPU hours. Genetic algorithms utilize a cost function, which defines the optimization goal, to effectively obtain near-optimal solutions while reducing the exploration time for a wide range of real-world optimization problems [44]. The framework uses genetic algorithms that rely on the concepts of reproduction and evolution to select the models that need to be trained in each generation to create a new generation of models that have the potential to further optimize the cost function. The use of other meta-heuristic approaches to explore the design space is orthogonal to our use of genetic algorithms, which encompasses techniques like ant colony optimization [8] and simulated annealing [48], and can be incorporated into the framework, if essential.

In the selective exploration process, we start with an initial population of 30 random DNN models present in the design space, referred to as individuals, based on the recommendation of previous works [41] in order to obtain the best results. The genetic algorithms require the presence of a "chromosome," which encodes all the key neural architecture parameters ("genes") that can be varied to obtain the complete design space of DNN models. All the *genes* are stitched together to generate the *chromosome* string, which, when decoded, constructs a DNN model, or an individual, in the design space. Each individual is subsequently trained on the constructed dataset to evaluate its viability in terms of a *fitness value*, which enables it to compete with other individuals in the design space. The fitness value is estimated as the cost function when the decoded DNN model (M) exists in the design space (ψ) or is considered to be a `NULL` value otherwise and is discarded from the search. Next, on the basis of their fitness values, two individuals are selected to pass on their genes to the next generation while undergoing the process of mutation and crossover, which are essential reproduction principles. We ensure an ordered 0.4 crossover probability for a mating parent pair with a random crossover location in the pair's chromosomes. The next generation of the population, i.e., the offspring, has their parents' chromosomes exchanged from the start until the crossover point and is considered for exploration based on their fitness value. The offspring also have a mutation probability of 0.11 to enable a bit-flip in the chromosome, thereby ensuring a diverse population and enabling a comprehensive exploration of DNN models. The experiments are run to determine a population of 30 individuals in each generation, based on their fitness values, to create 5 consecutive iterations of offspring that can be trained and evaluated to determine the set of best-fit individuals (see Fig. 3).

By default, the framework includes the ability to explore the design space using the following recognized genetic algorithms: NSGA-II [6], Roulette Wheel [11], Tournament Selection [31], and SPEA-2 [51]. Likewise, other algorithms and heuristics can be incorporated into the framework, as discussed earlier. The time complexity of each algorithm determines the order of execution time required for

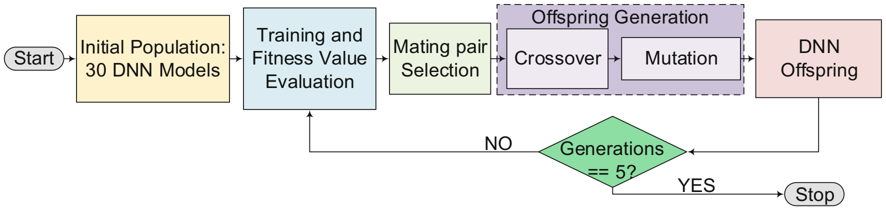

Fig. 3 Flow chart illustrating the selective design space exploration technique (adapted from [37])

exploring the design space ψ. If the size of the design space is considered to be N, the time complexity of the algorithms would be $O(N^2)$, $O(N * \log N)$, $O(N)$, and $O(N^2 * \log N)$, respectively. The efficacy of these algorithms, illustrated by the varying subset of individuals selected and evaluated, is discussed in Sect. 3 with the use-case. The genetic algorithms used by the framework require a cost function (ϕ) that needs to be specified by the system designer, which can be optimized to obtain the set of near-optimal network models (ω) for the explored design space. The weighted cost function used in this framework is

$$\phi = \alpha * Q + \beta * \left[1 - \frac{H}{H_{\max}}\right]$$

where $\alpha, \beta \in [0, 1]$ depict the weights for output quality (Q) and hardware requirements (H) of the model, respectively. $H_{\max}$ denotes the hardware requirements of the state-of-the-art baseline model. As discussed earlier, Q can be evaluated as Accuracy, Precision, Recall, or F1-score, whereas H can be estimated as the memory overhead or the number of floating-point operations for an inference. Other application-specific quality metrics or additional hardware requirements, such as the power consumption of the model or its energy requirements on the target platform, can also be included in the framework. The weights α and β depict the importance of the quality and hardware metrics, respectively, during the algorithm's exploration of the design space. Algorithm 1 discusses the pseudo-code for the weighted DNN model exploration technique deployed in the framework. Given (1) the inputs (design space (ψ), weights for the cost function (α, β), and the hardware requirement for the state-of-the-art model ($H_{\max}$)) and (2) quality and hardware constraints (Q_{Const}, H_{Const}), the weighted DNN model exploration algorithm generates a set of DNN models ω that satisfies the quality and hardware constraints of the application. The *ExplorationAlgorithm* function call in Line 10 can call any of the selective exploration algorithms (genetic algorithms) or the exhaustive exploration technique discussed earlier. Table 2 illustrates an overview of the symbols and denotations used in this chapter.

Algorithm 1 Weighted DNN model exploration

```
Input: $\psi, \alpha, \beta, H_{\max}$
Constraints: $Q_{Const}, H_{Const}$
Output: $\omega$
 1: $H = []$;
 2: for M in $\psi$ do
 3:   if $HardwareRequirements(M) \leq H_{Const}$ then
 4:     $H.append(HardwareRequirements(M))$;
 5:   else
 6:     $\psi.remove(M)$;
 7:   end if
 8: end for
 9: $\phi = \alpha * Q + \beta * \left[1 - \frac{H}{H_{\max}}\right]$
10: $\omega = ExplorationAlgorithm(\phi, \psi)$;
11: for DNN in $\omega$ do
12:   if $Q.(\text{DNN}) < Q_{Const}$ then
13:     $\omega.remove(\text{DNN})$;
14:   end if
15: end for
```

Table 2 Overview of the symbols used in this work along with their denotations [37]

Symbol	Denotation	Symbol	Denotation
Q	Model quality	N	Size of design space (ψ)
Q_{Const}	User quality constraint	ϕ	Cost function to be optimized
A	Accuracy of the model	ω	Output set of near-optimal DNN models
P	Precision of the model	α	Weight for output quality Q
R	Recall of the model	β	Weight for hardware requirement H
F	F1-score of the model	M	DNN model in ψ
B	Memory overhead of the model	H_{Const}	Platform's hardware constraint
FP	No. of floating-point operations reqd. by the model	$H_{\max}$	Hardware requirements of the state-of-the-art model
ψ	Design space of DNN models	H	Hardware requirements of the model

2.6 Model Compression

The framework also includes the capability of further reducing the model's hardware requirements through the means of compression techniques like pruning and quantization. Besides neural architecture search approaches, model compression techniques have proven to be highly successful in reducing the hardware requirements of the model while retaining output quality [12].

2.6.1 Pruning

As the name conveys, the core concept of this approach involves identifying less-important parameters of the model, such as the weights, kernels, biases, or even neurons or layers, and eliminating them to further reduce the hardware requirements of the DNN model, increasing their deployability in edge platforms. Eliminating the model parameters reduces many of its requirements, such as memory overhead of the model and the number of floating-point operations required for an inference, which tend to further improve performance and reduce energy consumption on the target platform during inference. The pruned model is subsequently retrained on the constructed dataset to ensure that the model achieves an output quality similar to that of the original unpruned model obtained from the design space. The framework integrates the pruning techniques presented in [3, 12, 26, 28, 30] to provide the system designer with a range of options that can be implemented in order to meet the application requirements based on the DNN model's capabilities. For example, the technique proposed in [12] determines the lowest $x\%$ of weights, based on their absolute magnitude, in each individual layer of the model and eliminates them, followed by a retraining stage, as discussed earlier, to achieve an accuracy similar to the original model. Whereas the technique presented in [30] sorts the complete set of weights in the model to iteratively eliminate the lowest $x\%$ of overall weights in each iteration, regardless of the layer, followed by model retraining to achieve original model accuracy. Section 3 illustrates an overview of the benefits of pruning DNN models obtained using this approach. Incorporating other pruning techniques into the framework can be easily achieved as long as the new technique complies with the original interfaces of standard pruning techniques.

2.6.2 Quantization

The model parameters are usually stored in a floating-point format requiring 32 bits, leading to a large memory overhead on the execution platform. Accessing each floating-point parameter from memory requires increased access latency and energy consumption, as opposed to traditional 8-bit or 16-bit integers. Likewise, a high-precision floating-point addition operation requires nearly an order of magnitude more energy as opposed to a 32-bit integer ADD operation [13]. Hence, approaches that can be used to reduce the precision from 32 bits to 16 or 8 bits, through the process of quantization, can be used to substantially reduce the hardware requirements of the model. Quantization techniques can be implemented to further reduce the precision of the DNN model to less than 8 bits, by analyzing its trade-off with output quality for the target application. The process involves the construction of 2^p clusters, where p stands for the number of quantized bits, using the k-means algorithm, which evaluates the parameters in each layer of the DNN model. Once the clusters are determined, equally spaced values are allocated to each cluster ranging from minimum to maximum value for corresponding cluster weights composed of all zeros to all ones, respectively. For simplicity, all layers in the DNN model are quantized with the same number of bits. Similar to pruning, other quantization techniques can be incorporated into the framework as long as the new technique complies with the original interfaces of standard quantization.

Based on recommendations from the studies presented in [12] and from exhaustive experimentation, the optimal approach for minimizing the hardware requirements of the model requires pruning the selected DNN model obtained from the design space, followed by model quantization, to eliminate the redundant parameters and subsequently reduce parameter precision, respectively.

3 Case Study: Bio-signal Anomaly Detection

We present the efficacy of the framework by deploying it to generate, explore, and compress a wide range of DNN models for our use-case: ECG Bio-signal processing. We explore 5 different sub-cases as a part of this study:

- **UC**$_1$: Binary Classification: [Normal, Anomaly]
- **UC**$_2$: Multi-class Classification:
  [Normal, Premature Ventricular Contraction, Other Anomaly]
- **UC**$_3$: Multi-class Classification: [Normal, Bundle Branch Block, Other Anomaly]
- **UC**$_4$: Multi-class Classification:
  [Normal, Atrial Anomaly, Ventricular Anomaly, Other Anomaly]
- **UC**$_5$: Multi-class Classification: [Normal, Ventricular Fibrillation, Other Anomaly]

Hannun et al. [14] proposed a deep neural network model architecture that can differentiate between 12 classes of ECG signals, evaluated on their private dataset. This model is considered to be the current state of the art in ECG signal classification and is the baseline model of this use-case. The primary block used in [14] is adopted in this use-case to generate the design space of DNN models for each of the 5 different sub-cases discussed above. The input and output layers have been modified to consider the data from the open-source ECG dataset adopted in this case study to process and categorize them into the required output classes. The default model of the DNN is modified to include LSTM cells at the end, enabling accuracy improvements in cases where the number of feature extraction layers is substantially reduced during neural architecture search.

3.1 Experimental Setup

Dataset Construction For this bio-signal processing case study, the MIT-BIH dataset [32] is used to construct the required datasets by collecting a 256-sample window, which is subsequently assigned a label corresponding to the original labels of the parent dataset. The 41 different annotations of the parent dataset are categorized as one of the labels for each sub-case to ensure coherence in the dataset. To construct an enriched dataset that can provide the relevant information to the DNN model and enable it to learn effectively across labels like ventricular tachycardia and ventricular fibrillation, the framework also includes the CU Ventricular dataset [33] during the construction of the custom datasets. The constructed datasets are split in the ratio of 7:1:2 to generate the training, validation, and testing datasets, respectively.

Neural Architecture Parameters An overview of the modified DNN architecture used in this case study is presented in Fig. 4. Therefore, the three primary neural architecture parameters that can be varied to generate the DNN model design space are (1) #ResNet Blocks, (2) #Filters, and (3) #LSTM Cells. The ResNet blocks are made of 1D convolutional layers, batch normalization, ReLU activation blocks, and dropout layers, as illustrated in Fig. 4, and can vary between 0 and 15. The number of filters, of size 16, in each convolution layer is determined as a function of z–$[32 \times 2^z]$—where z starts from the value of 0 and is increased by 1 after every y ResNet blocks (y varies from 1 to 4 in increments of 1, i.e., $y \in \{1, 2, 3, 4\}$). The number of LSTM cells is varied as 2^x, where $x \in \{4, 5, 6, 7, 8\}$.

By varying these parameters, we can generate up to 320 different DNN models as part of a given application's design space. However, due to the hardware limitation imposed by the state-of-the-art model, the framework reduces the number of models explored to 135, thereby drastically reducing the exploration time.

Selective Exploration Figure 5 presents the composition of the chromosome used by the genetic algorithms in this case study. The chromosome is a binary string of size 9, which encodes the key neural architecture parameters discussed

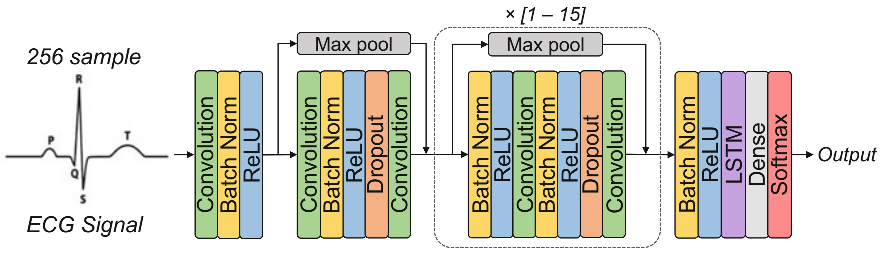

Fig. 4 Modified state-of-the-art DNN architecture used in the case study (adapted from [37])

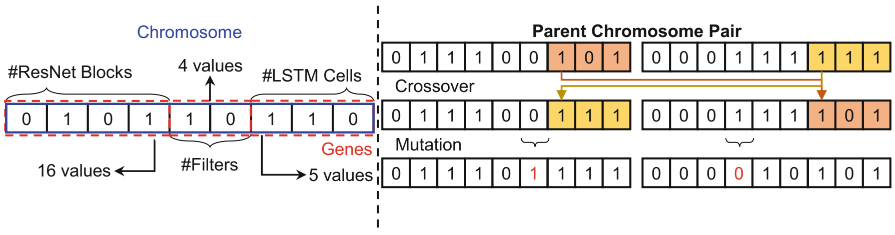

Fig. 5 The composition of the chromosome used by the genetic algorithms in this case study; example of chromosomal crossover and mutation (adapted from [37])

Table 3 Optimal hyper-parameter values used for training the DNN Models [37]

Hyper-parameter	Optimal value
Weights initialization	He et al. [16]
Adam optimizer [22]	$\beta_1 = 0.9$, $\beta_2 = 0.999$
Learning rate	0.001
Batch size	128
Dropout	0.2

Table 4 The optimal values of the constants used by the genetic algorithms [37]

Constant	Optimal value
Population size	30
Chromosome length	9
Generation size	5
Mutation probability	0.11
Crossover probability	0.4

above as genes. The chromosome can therefore construct $2^{10} - 1$, or 1023, DNN models in design space for each of the sub-cases. However, since only 5 of the 7 possible #LSTM cell values lead to valid DNN model architectures, we can directly eliminate invalid configurations not present in ψ. Once the parent chromosome pair is selected, based on their fitness value, for generating offspring, they undergo the process of crossover to exchange genes and undergo potential mutation to introduce diversity, as illustrated in Fig. 5.

Tool Flow The TensorFlow platform is used for the implementation of the DNN models in the Python programming environment with the help of the Keras package. The DNN models are trained over multiple iterations with varying hyper-parameter values to determine the ones that offer maximum accuracy. Table 3 presents the optimal values of these hyper-parameters. The DEAP library [5] in Python contains implementations of the four genetic algorithms that are used in the case study. Table 4 presents the constants and their optimal values, which are used by the genetic algorithms during selective exploration of the design space. The exploration stage is executed on a GPU server composed of four i9 CPUs and 8 Nvidia RTX 2080 GPUs, with the early stopping mechanism enabled. The selected models are then trained using the custom dataset for quality evaluation and studying the trade-off with their hardware requirement.

3.2 Exhaustive Exploration

Figure 6 illustrates the results and trade-offs between quality and memory of exhaustively exploring the models in the UC_3 design space. The Pareto-frontier of the complete design space, which connects all the Pareto-optimal DNN models and offers the best trade-off between quality and memory, is illustrated by Ⓐ. Label Ⓑ,

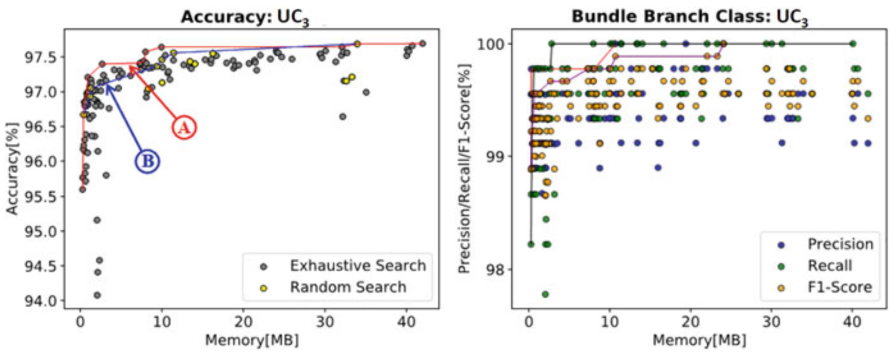

Fig. 6 Analysis of exhaustive exploration on the UC_3 design space (adapted from [37])

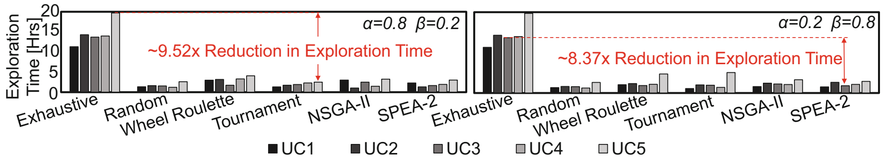

Fig. 7 Analyzing the time benefits of the selective exploration approach (adapted from [37])

on the other hand, depicts the pseudo-Pareto-frontier constructed using the set of optimal designs obtained by random exploration (i.e., baseline). The large number of inter-dependent parameters in DNN models leads to the situation where the designs depicted in Ⓐ and Ⓑ are very similar to each other. Exhaustive exploration of the design space has led to the successful identification of a DNN model that can reduce the overhead by ∼30 MB for a quality loss of less than 0.5%. However, due to the time required for such exhaustive exploration, it might be more suitable to obtain a near-optimal point that offers similar trade-offs using selective exploration for much less time. The variance in the Precision, Recall, and F1-score of the model for the specialized bundle branch class indicates suitability of the framework to impose a quality constraint on these metrics as well.

3.3 Selective Exploration: Time Benefits

The primary benefit of the selective exploration process is the reduction in time required to search the design space of DNN models with the use of genetic algorithms. Figure 7 illustrates the reduction in time for the five different use-cases when explored using the genetic algorithms, as opposed to exhaustive exploration. We have also varied the weights used by the cost function (α, β) to emphasize that changing weights does not drastically modify the time needed for exploring the design space. Randomly selecting and evaluating 10% of the DNN models

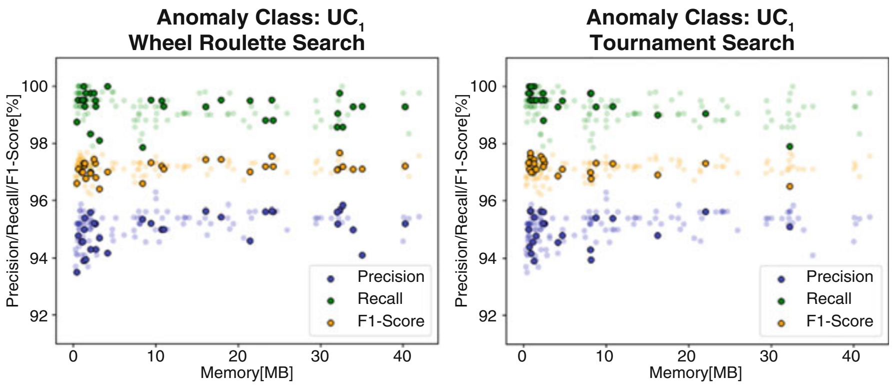

Fig. 8 Evaluation of the quality and memory trade-offs for the models obtained using wheel roulette search and tournament search on the UC_1 design space (adapted from [37])

in the design space acts as baseline comparison for the selective exploration strategies discussed in this chapter, with practically no algorithmic overhead. The selective exploration strategies achieve $9\times$ reduction in exploration time, on average, as opposed to the bounded exhaustive exploration strategy. The use of genetic algorithms for exploring the search space is highly beneficial in scenarios where the application requires the use of highly complex deep neural networks with tens of millions of parameters. Exhaustively training and evaluating each model in the design space, in such instances, would lead to exploration time overheads of hundreds of GPU hours, which might not be feasible for the system designer.

3.4 Selective Exploration: Efficacy and Analysis

The primary benefit of using genetic algorithms, reduction in exploration time, was already discussed earlier. In this subsection, we focus on the capability of the genetic algorithms in exploring the design space and analyze their efficacy. The results of these experiments for the UC_1 and UC_5 design spaces, with α and β set to 0.5 are illustrated in Figs. 8 and 9, respectively. The transparent points in these results depict the models obtained from the design space using exhaustive exploration, enabling us to determine the efficacy of the genetic algorithms. The genetic algorithms are highly successful at identifying a set of near-optimal DNN models without traversing the complete design space, especially in cases where the accuracy improvements or the hardware memory reductions are minimal when compared to the Pareto-optimal design. The number of models evaluated by the NSGA-II and SPEA-2 algorithm is smaller than their counterparts. Note that a significant number of models in the UC_5 design space exhibit 0% quality due to

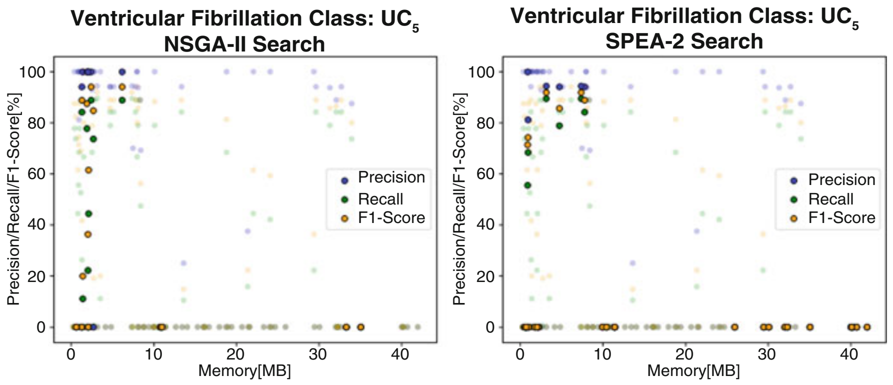

Fig. 9 Evaluation of the quality and memory trade-offs for the models obtained using NSGA-II search and SPEA-2 search on the UC_5 design space (adapted from [37])

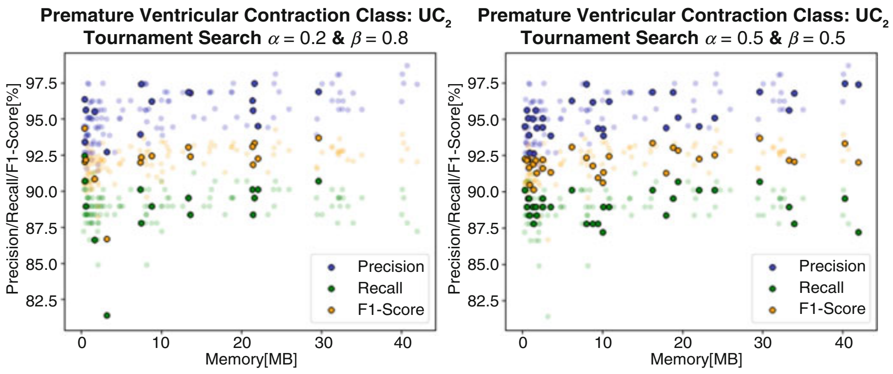

Fig. 10 Evaluation of the weighted exploration technique on the UC_2 design space using tournament search with two different weight values for the cost function (adapted from [37])

the inherent differences in the number of samples of class Ventricular Fibrillation, leading to a bias against it.

3.5 Selective Exploration: Weighted Exploration

Next, we discuss a subset of the results obtained when exploring the design space using different weights for the cost function (ϕ), which is used by the genetic algorithms. Figure 10 illustrates the results when the algorithm focuses on optimizing (1) memory alone ($\alpha = 0.2, \beta = 0.8$) or (2) memory and accuracy ($\alpha = 0.5, \beta = 0.5$), for a model in the design space of UC_2. Similarly, Fig. 11 presents the results when the algorithm optimizes for (1) memory alone ($\alpha = 0.2, \beta = 0.8$) or (2) accuracy alone ($\alpha = 0.5, \beta = 0.5$), in the design space of

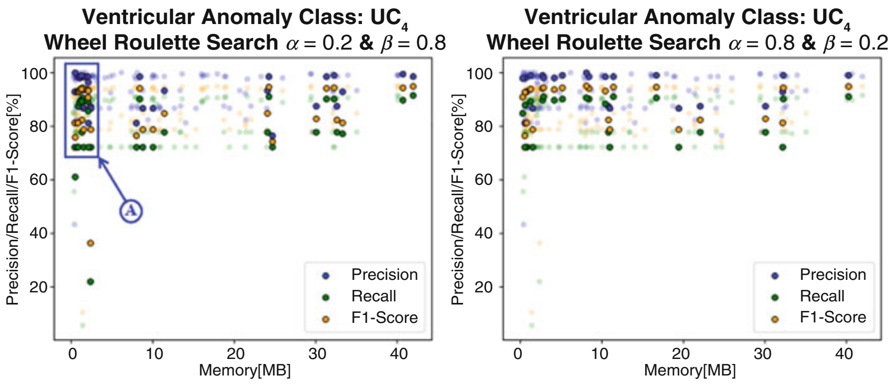

Fig. 11 Evaluation of the weighted exploration technique on the UC$_4$ design space using wheel roulette search with two different weight values for the cost function (adapted from [37])

UC$_4$. The weighted parameterization of the cost function is highly beneficial in guiding the genetic algorithms to optimize for the required parameter, i.e., memory or quality or both. For example, as illustrated by Ⓐ in Fig. 11, the algorithm focuses on optimizing memory, thereby selecting a large number of points with minimal overhead. Similarly, when the optimization goal is either accuracy only (see Fig. 11) or memory and accuracy (see Fig. 10), appropriate models are selected for evaluation by the algorithm.

3.6 Pruning and Quantization: Compression Efficacy and Receiver Operating Characteristics

Next, we select three near-optimal models obtained from the UC$_4$ design space to evaluate the efficacy of our pruning and quantization techniques. Without loss of generality and for the purpose of illustration, the three models, Z_1, Z_2, and Z_3, focus on accuracy alone, trade-off between accuracy and memory, or memory alone, respectively. Pruning, alone, is quite effective in reducing the memory by nearly 40% for roughly 0.15% increase in accuracy. This contra-indicative improvement in accuracy can be attributed to the over-redundant parameterization of the network model, which is eliminated by pruning. Due to similar reasons, model Z_1 can tolerate pruning of a significant percentage of parameters before exhibiting accuracy losses, as opposed to the other models that are not as over-parameterized. Likewise, quantization can drastically reduce the memory requirements of the network by lowering the precision of the parameters storing the weights and biases. This process can further reduce the memory requirements by up to 5×, as opposed to FP32 precision, for <0.1% quality loss. Combining both these approaches can reduce the memory by a factor of 53× for <0.2% loss in quality (Fig. 12).

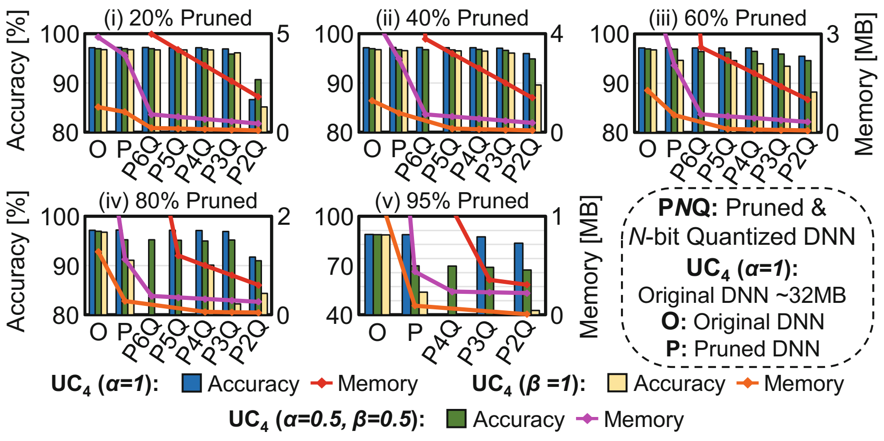

Fig. 12 Compression of the near-optimal UC$_4$ design space DNN models (adapted from [37])

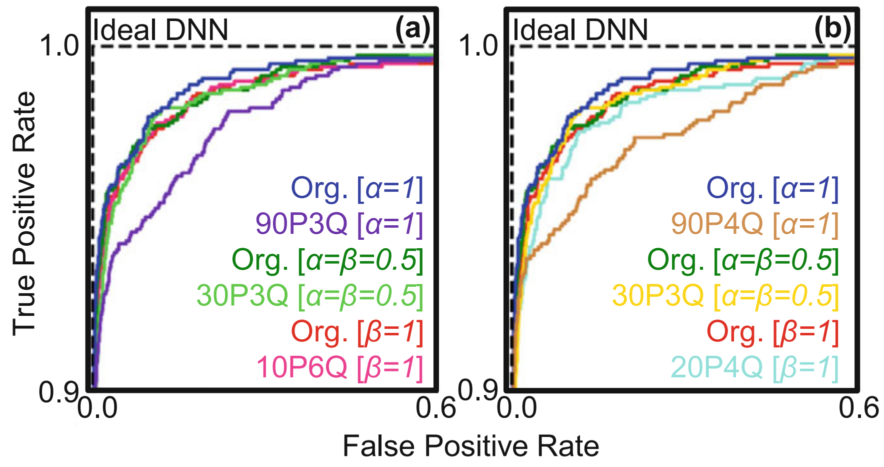

Fig. 13 ROC evaluation of the similar DNN models from the UC$_4$ design space [37]

Next, the receiver operating characteristics of selected pruned and quantized models are evaluated to determine their behavior, when compared to the original model obtained from the design space. The evaluation is completed for two scenarios:

(a) With a memory constraint of 0.5MB, maximize the model accuracy.
(b) With an accuracy constraint of 96.7%, minimize the model's memory.

As can be observed from the results presented in Fig. 13, model Z_1 exhibits the best operating characteristics, with Z_2 and Z_3 not lagging far behind. The maximum accuracy models, which are subsequently pruned and quantized, exhibit the worst operating characteristics, far behind the pruned and quantized models of Z_2 and Z_3, albeit with similar accuracy metrics. This makes the latter two models more suitable for deployment in the real world on constrained edge devices like wearables.

4 Conclusion

Healthcare is one of the world's largest industries requiring a lot of investments, man power, training, and expertise, especially with the rising older population (above the age of 65) in most western nations, a significant percentage of whom require continuous support and healthcare. This requires scientists and researchers to develop technologies that can cater to the requirements of global healthcare systems with currently available technologies. Deep learning, which is currently at the forefront of major technological innovation, has proven to be highly effective in various healthcare domains like medical imaging, electronic health data analytics, precision medicine, and drug discovery. In this chapter, an embedded neural architecture search and model compression framework was discussed to enable their deployment in healthcare applications. The framework considers the user requirements, in terms of quality, or specifications, like type of output, and hardware constraints of the target platform to effectively search the design space of DNN models to generate a set of near-optimal DNNs suitable for the application. Besides achieving a 53× reduction in memory, models optimized for both accuracy and memory during the design space search were observed to have better operating characteristics, even when compressed. The framework is open source and available online at https://bionetexplorer.sourceforge.io/.

Acknowledgments This work has been supported by the Doctoral College Resilient Embedded Systems, which is run jointly by the TU Wien's Faculty of Informatics and the UAS Technikum Wien. We would also like to acknowledge the coauthors (collaboration partners) of our BioNetExplorer journal paper, which is discussed in this chapter.

References

1. Aggarwal, R., Sounderajah, V., Martin, G., Ting, D.S., Karthikesalingam, A., King, D., Ashrafian, H., Darzi, A.: Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis. NPJ Digit. Med. **4**(1), 1–23 (2021)
2. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutorials **17**(4), 2347–2376 (2015)
3. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM J. Emerg. Technol. Comput. Syst. **13**(3), 1–18 (2017)
4. Bloomberg: These are the economies with the most (and least) efficient health care. [Online Link]
5. De Rainville, F.M., Fortin, F.A., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: A python framework for evolutionary algorithms. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 85–92 (2012)

6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-ii. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
7. Dincer, A.B., Celik, S., Hiranuma, N., Lee, S.I.: DeepProfile: Deep learning of cancer molecular profiles for precision medicine. BioRxiv, p. 278739 (2018)
8. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006)
9. Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., Dean, J.: A guide to deep learning in healthcare. Nat. Med. **25**(1), 24–29 (2019)
10. Gibson, E., Li, W., Sudre, C., Fidon, L., Shakir, D.I., Wang, G., Eaton-Rosen, Z., Gray, R., Doel, T., Hu, Y., et al.: NiftyNet: a deep-learning platform for medical imaging. Comput. Methods Programs Biomed. **158**, 113–122 (2018)
11. Goldberg, D.E.: Optimization, and machine learning. Genetic algorithms in Search (1989)
12. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint arXiv:1510.00149 (2015)
13. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, vol. 28 (2015)
14. Hannun, A.Y., Rajpurkar, P., Haghpanahi, M., Tison, G.H., Bourn, C., Turakhia, M.P., Ng, A.Y.: Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nat. Med. **25**(1), 65–69 (2019)
15. Hassaballah, M., Awad, A.I.: Deep Learning in Computer Vision: Principles and Applications. CRC Press, Boca Raton (2020)
16. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034 (2015)
17. Horst, F., Lapuschkin, S., Samek, W., Müller, K.R., Schöllhorn, W.I.: Explaining the unique nature of individual gait patterns with deep learning. Sci. Rep. **9**(1), 1–13 (2019)
18. Huang, S.C., Pareek, A., Seyyedi, S., Banerjee, I., Lungren, M.P.: Fusion of medical imaging and electronic health records using deep learning: a systematic review and implementation guidelines. NPJ Digit. Med. **3**(1), 1–9 (2020)
19. Islam, S.R., Kwak, D., Kabir, M.H., Hossain, M., Kwak, K.S.: The internet of things for health care: a comprehensive survey. IEEE Access **3**, 678–708 (2015)
20. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al.: Highly accurate protein structure prediction with AlphaFold. Nature **596**(7873), 583–589 (2021)
21. Kamath, U., Liu, J., Whitaker, J.: Deep Learning for NLP and Speech Recognition, vol. 84. Springer, Berlin (2019)
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014). arXiv preprint arXiv:1412.6980
23. Korkalainen, H., Aakko, J., Nikkonen, S., Kainulainen, S., Leino, A., Duce, B., Afara, I.O., Myllymaa, S., Töyräs, J., Leppänen, T.: Accurate deep learning-based sleep staging in a clinical population with suspected obstructive sleep apnea. IEEE J. Biomed. Health Inform. **24**(7), 2073–2081 (2019)
24. Lavecchia, A.: Deep learning in drug discovery: opportunities, challenges and future prospects. Drug Discovery Today **24**(10), 2017–2032 (2019)
25. Lee, J.G., Jun, S., Cho, Y.W., Lee, H., Kim, G.B., Seo, J.B., Kim, N.: Deep learning in medical imaging: general overview. Korean J. Radiol. **18**(4), 570–584 (2017)
26. Lin, J., Rao, Y., Lu, J., Zhou, J.: Runtime neural pruning. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
27. Lundervold, A.S., Lundervold, A.: An overview of deep learning in medical imaging focusing on MRI. Zeitschrift für Medizinische Physik **29**(2), 102–127 (2019)
28. Luo, J.H., Wu, J., Lin, W.: ThiNet: A filter level pruning method for deep neural network compression. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 5058–5066 (2017)

29. Manyika, J., Chui, M., Bughin, J., Dobbs, R., Bisson, P., Marrs, A.: Disruptive technologies: advances that will transform life, business, and the global economy, vol. 180. McKinsey Global Institute San Francisco (2013)
30. Marchisio, A., Hanif, M.A., Martina, M., Shafique, M.: PruNet: Class-blind pruning method for deep neural networks. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2018)
31. Miller, B.L., Goldberg, D.E., et al.: Genetic algorithms, tournament selection, and the effects of noise. Complex Syst. **9**(3), 193–212 (1995)
32. Moody, G.B., Mark, R.G.: The impact of the MIT-BIH arrhythmia database. IEEE Eng. Med. Biol. Mag. **20**(3), 45–50 (2001)
33. Nolle, F., Badura, F., Catlett, J., Bowser, R., Sketch, M.: CREI-GARD, a new concept in computerized arrhythmia monitoring systems. Comput. Cardiol. **13**, 515–518 (1986)
34. Our World in Data: Life expectancy. [Online Link]
35. Policy Advice: The state of healthcare industry—statistics for 2021. [Online Link]
36. Porumb, M., Stranges, S., Pescapè, A., Pecchia, L.: Precision medicine and artificial intelligence: a pilot study on deep learning for hypoglycemic events detection based on ECG. Sci. Rep. **10**(1), 1–16 (2020)
37. Prabakaran, B.S., Akhtar, A., Rehman, S., Hasan, O., Shafique, M.: BioNetExplorer: architecture-space exploration of biosignal processing deep neural networks for wearables. IEEE Internet Things J. **8**(17), 13251–13265 (2021)
38. Preuer, K., Klambauer, G., Rippmann, F., Hochreiter, S., Unterthiner, T.: Interpretable deep learning in drug discovery. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, pp. 331–345. Springer, Berlin (2019)
39. Rajkomar, A., Oren, E., Chen, K., Dai, A.M., Hajaj, N., Hardt, M., Liu, P.J., Liu, X., Marcus, J., Sun, M., et al.: Scalable and accurate deep learning with electronic health records. NPJ Digit. Med. **1**(1), 1–10 (2018)
40. Ramsundar, B., Eastman, P., Walters, P., Pande, V.: Deep Learning for the Life Sciences: Applying Deep Learning to Genomics, Microscopy, Drug Discovery, and More. O'Reilly Media (2019)
41. Reeves, C., Rowe, J.E.: Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory, vol. 20. Springer, Berlin (2002)
42. Rifaioglu, A.S., Atas, H., Martin, M.J., Cetin-Atalay, R., Atalay, V., Doğan, T.: Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases. Briefings Bioinform. **20**(5), 1878–1912 (2019)
43. Sahiner, B., Pezeshk, A., Hadjiiski, L.M., Wang, X., Drukker, K., Cha, K.H., Summers, R.M., Giger, M.L.: Deep learning in medical imaging and radiation therapy. Med. Phys. **46**(1), e1–e36 (2019)
44. Sastry, K., Goldberg, D., Kendall, G.: Genetic algorithms. In: Search Methodologies, pp. 97–125. Springer, Berlin (2005)
45. Shickel, B., Tighe, P.J., Bihorac, A., Rashidi, P.: Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. IEEE J. Biomed. Health Inform. **22**(5), 1589–1604 (2017)
46. Solares, J.R.A., Raimondi, F.E.D., Zhu, Y., Rahimian, F., Canoy, D., Tran, J., Gomes, A.C.P., Payberah, A.H., Zottoli, M., Nazarzadeh, M., et al.: Deep learning for electronic health records: a comparative review of multiple deep neural architectures. J. Biomed. Inform. **101**, 103337 (2020)
47. United Nations: World population ageing. [Online Link]
48. Van Laarhoven, P.J., Aarts, E.H.: Simulated annealing. In: Simulated Annealing: Theory and Applications, pp. 7–15. Springer, Berlin (1987)
49. Wang, F., Casalino, L.P., Khullar, D.: Deep learning in medicine—promise, progress, and challenges. JAMA Internal Med. **179**(3), 293–294 (2019)

50. Xiao, C., Choi, E., Sun, J.: Opportunities and challenges in developing deep learning models using electronic health records data: a systematic review. J. Am. Med. Inform. Assoc. **25**(10), 1419–1428 (2018)
51. Zitzler, E., Laumanns, M., Thiele, L.: Spea2: improving the strength pareto evolutionary algorithm. In: TIK-Report, vol. 103 (2001)