# D3.2

# Initial Approaches for Decentralised and High-Performance Learning

| | |
|---|---|
| **Project Title** | AI4Media - A European Excellence Centre for Media, Society and Democracy |
| **Contract No.** | 951911 |
| **Instrument** | Research and Innovation Action |
| **Thematic Priority** | H2020-EU.2.1.1. - INDUSTRIAL LEADERSHIP - Leadership in enabling and industrial technologies - Information and Communication Technologies (ICT) / ICT-48-2020 - Towards a vibrant European network of AI excellence centres |
| **Start of Project** | 1 September 2020 |
| **Duration** | 48 months |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951911

| | |
|---|---|
| Deliverable title | Initial Approaches for Decentralised and High-Performance Learning |
| Deliverable number | D3.2 |
| Deliverable version | 1.0 |
| Previous version(s) | N/A |
| Contractual date of delivery | 30 November 2021 |
| Actual date of delivery | 3 December 2021 |
| Deliverable filename | AI4Media_D3_2-final.pdf |
| Nature of deliverable | Report |
| Dissemination level | Public |
| Number of pages | 52 |
| Work Package | WP3 |
| Task(s) | T3.4, T3.5, T3.8 |
| Parner responsible | CERTH |
| Author(s) | Nikolaos Giatsoglou (CERTH), Symeon Papadopoulos (CERTH), Ioannis Kompatsiaris (CERTH), Nicu Sebe (UNITN), Hannes Fassold (JR), François Schnitzler (IDF), Martin Wistuba (IBM), Killian Levacher (IBM), Artur Garcia Saez (BSC) |
| Editor | Symeon Papadopoulos (CERTH) |
| Officer | Evangelia Markidou |

| | |
|---|---|
| Abstract | This document presents initial outcomes of the research on decentralized and high-performance learning in WP3 of AI4Media. As such, it outlines the research advances of contributing partners in T3.4, T3.5, and T3.8 at month 15. For each task, we present partner contributions, relevant publications and links to developed software (if available). Finally, it discusses plans for ongoing activities and future research. |
| Keywords | AI, Machine Learning, Architecture Search, Optimization, Decentralization, Distributed Training, Resource Adaptation, Quantum Learning |

www.ai4media.eu

info@ai4media.eu

## Copyright
© Copyright 2021 AI4Media Consortium

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the AI4Media Consortium. In addition to such written permission to copy, reproduce, or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

All rights reserved.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951911

## Contributors

| NAME | ORGANIZATION |
|---|---|
| Nikolaos Giatsoglou | CERTH |
| Symeon Papadopoulos | CERTH |
| Ioannis Kompatsiaris | CERTH |
| Nicu Sebe | UNITN |
| Hannes Fassold | JR |
| François Schnitzler | IDF |
| Martin Wistuba | IBM |
| Killian Levacher | IBM |
| Artur Garcia Saez | BSC |

## Peer Reviews

| NAME | ORGANIZATION |
|---|---|
| Ioannis Mademlis | AUTH |
| Mihai Gabriel Constantin | UPB |

## Revision History

| Version | Date | Reviewer | Modifications |
|---|---|---|---|
| 0.1 | 13/9/2021 | Nikolaos Giatsoglou, Symeon Papadopoulos | First draft sent to partners for contributions |
| 0.2 | 25/10/2021 | Nikolaos Giatsoglou, Symeon Papadopoulos | Updated version including inputs from CERTH, UNITN, JR, IDF, IBM and BSC in sections 3,4 and 5. |
| 0.3 | 5/11/2021 | Nikolaos Giatsoglou, Symeon Papadopoulos, Ioannis Kompatsiaris | Updated version including inputs for the executive summary and task introductions in sections 3, 4 and 5 by CERTH, UNITN and BSC. |
| 0.4 | 12/11/2021 | Nikolaos Giatsoglou, Symeon Papadopoulos | Updated version including inputs for sections 2 and 6 by CERTH, UNITN, JR, IDF, IBM and BSC. |
| 0.5 | 28/11/2021 | Nikolaos Giatsoglou, Symeon Papadopoulos, Filareti Tsalakanidou | Updated version and quality management based on internal review comments and replies to them by CERTH, UNITN, JR, IBM and BSC. |
| 1.0 | 30/11/2021 | Nikolaos Giatsoglou, Symeon Papadopoulos, Ioannis Kompatsiaris | Final version |

The information and views set out in this report are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf.

## Table of Abbreviations and Acronyms

| Abbreviation | Meaning |
|---|---|
| AI | Artificial Intelligence |
| APoZ | Average Percentage of Zeros |
| BNN | Binarized Neural Network |
| BWN | Binary Weight Network |
| CNN | Convolutional Neural Network |
| CP | Canonical Polyadic |
| D | Deliverable |
| DNN | Deep Neural Network |
| FSBO | Few-Shot Bayesian Optimization |
| GDPR | European General Data Protection |
| GNN | Graph Neural Network |
| GP | Gaussian Process |
| GPU | Graphics Processing Unit |
| HAR | Human Activity Recognition |
| HPO | Hyperparameter Optimisation |
| IPKD | InPlace Knowledge Distillation |
| IID | Independent and Identically Distributed |
| KD | Knowledge Distillation |
| KL | Kullback-Leibler |
| LHS | Latin Hypercube Sampling |
| MC | Monte-Carlo |
| NAS | Neural Architecture Search |
| NLP | Natural Language Processing |
| ReLU | Rectified Linear Unit |
| RNN | Recurrent Neural Network |
| SGA | Stochastic Gradient Ascent |
| SGD | Stochastic Gradient Descent |
| SkipW | Skip-Windows |
| SVD | Singular Value Decomposition |
| T | Task |
| TA | Teacher Assistant |
| TWN | Ternary Weight Network |
| TTQ | Trained Ternary Quantization |
| UAT | Universal Approximation Theorem |
| WP | Work Package |
| WS | Warm Start |

## Contents

## List of Tables

## List of Figures

# 1. Executive Summary

This deliverable provides the research results of the activities in Task 3.4 (Neural Architecture Search), Task 3.5 (AI at the Edge, Decentralised and Distributed Learning) and Task 3.8 (Quantum Assisted Reinforcement Learning) at M15. We detail the motivation, new methods and results obtained by involved partners and, when available, provide explicit references to publications and developed software. We also position contributions with respect to WP8 use cases outlined in D8.1 *"Use Cases Definitions and Requirements"*.

This work reflects the progress of these tasks so far. All task participants have been active, many results have been successfully published in top venues of relevant communities and new software has been made publicly available. All tasks provide support for the implementation of Artificial Intelligence (AI) methodologies that can be used to support existing AI-powered services for media-centric applications, especially when dealing with large-scale data either through efficient algorithms (e.g., neural architecture search) or by running on emerging computing infrastructures (e.g., edge devices and quantum computers). We next present a summary of each task's initial approaches and ongoing research, as well as prospective directions.

**Neural architecture search (Task 3.4).** Contributions in this task include: (a) an efficient deep learning approach using model compression and acceleration, and (b) rethinking hyperparameter optimization as a few-shot learning task of surrogate models for parameter estimation. These mitigate the time needed to find high-performing neural architectures that best match available data by respectively reducing the run time of neural models and intelligently moving towards comparing few near-best hyperparameters. Thus, they enable replacement of human-driven hyperparameter exploration and its associated costs with automated processes that can be employed by non-experts.

Ongoing and future work for this task includes: (i) investigating neuro-evolutionary network architecture search approaches, which automatically search for optimal neural network architectures using evolutionary algorithms, and (ii) investigating efficient evolutionary algorithms with low memory footprints to evolve a dynamic image classifier.

**AI at the edge, decentralized and distributed learning (Task 3.5).** Contributions in this task include: (a) edge diffusion-based mining and learning in decentralized peer-to-peer networks, including adaptations of graph neural network breakthroughs for the classification of peer nodes based on local media content, (b) novel training strategies that improve model generalization (like mini-batch trimming), which will be subsequently ported to distributed training frameworks, and (c) resource-aware learning that includes lightweight recurrent neural networks and in-place knowledge distillation that trains deep learning architectures of progressively fewer parameters given the outputs of previous ones.

Ongoing and future work for this task includes: (i) decentralized learning without homophilous assumptions and peer-to-peer learning models of higher accuracy, (ii) new adaptive optimization methods for normal and distributed learning (*AdaFamily*), and (iii) inference under limited resources at the edge.

**Quantum assisted reinforcement learning (Task 3.8).** Contributions in this task consist of developing comprehensive theoretical tools that describe usage of single qubits, i.e., the quantum mechanics analogue to binary bits, as universal approximators of (two) bounded functions of any number of variables. This way, theoretical properties that support usage of neural networks to learn any kind of real-life processes can also be used to justify usage of quantum computing while also boasting maximal information compression and, therefore, maximal learning speedup. This is verified under real-world conditions by experimenting on a superconducting qubit, where traditional approximation of arbitrary mathematical functions is replicated using a hybrid Quantum computer operated by a classical optimization algorithm.

Ongoing and future work for this task includes the application of these techniques to create more complex operations, including the implementation of learning processes over classical data, and the execution of reinforcement learning tasks.

# 2. Introduction

The rapidly expanding volumes of data available for training AI models and systems increase training, execution and human investigation times with rates that the growth of technological resources struggles to cope with. For instance, billions of mobile devices in the wild have the capacity to generate proportional amounts of data as they are used by vibrant communities to produce and share all types of media, such as text, images, video, sound and social connections. Therefore, due to the sheer number of samples to be analysed, data processing can be time consuming, even with the support of Graphics Processing Unit (GPU) acceleration. Additionally, AI models are often highly parameterized (e.g., in terms of the number and breadth of neural layers or training dropout rates) and sensitive to small perturbations of their hyperparameters; tuning the latter often requires many training repetitions that multiply the cost of learning.

The above concerns have motivated the introduction of AI paradigms that exhibit a high degree of automation and computational efficiency when deployed in the real world. These can help avoid human-oriented risks, such as expert failure in immediately grasping optimal neural architectures, and can also minimize learning and inference times, for example with more efficient algorithms or new computing paradigms. In WP3, three promising directions pertaining to this objective are investigated, each of which aims to speed up different aspects of learning.

First, Task 3.4 addresses the concept of neural architecture search. Typically, the search for suitable architectures is a time-consuming, arduous and error-prone task. Thus, there are practical benefits in mitigating the time spent on manually selecting architectures (e.g. the time spent on investigating hyperparameter perturbations) by reducing the number of experiments run on new datasets. To do so, research focused on transfer neural architecture search, which leverages past system experiences on different datasets to accelerate search when new data are provided. This tackles a well-known problem of traditional architecture search approaches; that they start from scratch for new datasets. Focus was placed on producing new methods that leverage parameter sharing and differentiable architecture search and exploring these in terms of their applicability on multi-objective search and tasks beyond image and text classification. Research and outcomes of this task are presented in Section 3.

Second, Task 3.5 addresses issues of scale by proposing methods for AI computations to take place across multiple devices, by investigating three sub-directions: (a) decentralized AI in which fragments of inference models reside across multiple edge devices (e.g. mobile devices) and these collaborate to learn tight approximations of centralized equivalents, even under uncertain availability, (b) novel training strategies that improve model generalization (*mini-batch trimming*), which will be subsequently ported to distributed training frameworks and (c) reducing the size of pre-trained models so that these can be deployed on edge devices and run under resource constraints without the need for a central infrastructure. In all cases, AI systems scale with technological progress, given that new edge devices generating data or AI queries can either be the ones performing additional computations or also lead to the introduction of new cloud infrastructure that contributes to distributed training of gathered data. Research and outcomes of this task are presented in Section 4.

Finally, Task 3.8 aims to accelerate learning algorithms themselves by taking advantage of quantum theory breakthroughs that have the potential to rapidly reduce running time complexity. In detail, it investigates the usage of qubits as universal approximators that considerably speed up neural learning operations. Research and outcomes of this task are presented in Section 5.

# 3. Neural Architecture Search (Task 3.4)

## 3.1. Overview of our neural architecture search contributions

Deep learning models are becoming indispensable tools in many industries. Implementing them, however, requires a high level of expertise in neural network architectures. Additionally, the architectures one would select tend to vary significantly, depending on data domains and target AI tasks. To make matters worse, once adequate architectures are selected, making the best architectural choices, such as hyperparameter training process selection, is a tedious task that relies on trial-and-error. In particular, experts need to rely on their past experiences, technical expertise and understanding of the target application domain. Finally, evaluating the performance of such models is typically reduced to experts in the selected area in which the model will operate. For all these reasons, the need to automate as much as possible the processes involved in building deep learning neural networks is critical in order to fully democratise access to such technology across all industries. In this section, we present the work carried out so far in this task.

**UNITN** organized a workshop on Neural Architecture (NAS) search in conjunction with the CVPR 2021 conference and brought together emerging research areas such as automatic architecture search and hyperparameter optimization among others in order to discuss open challenges and opportunities ahead in each field. A special issue of the Pattern Recognition journal was also organized focusing on Deep Learning for Efficient and Robust Pattern Recognition, which explored NAS related topics such as model compression and acceleration. A survey on recent developments in deep learning was carried out, in which techniques that enable model compression and acceleration are explored.

**IBM** presents novel techniques addressing the need to reduce the trial-and-error processes of identifying optimal hyperparameters for given neural network architectures. Hyperparameter optimization (HPO) has so far mainly been performed using Bayesian optimization. This involves a parametric surrogate which is learned to approximate the black box response function of individual neural architectures. Unfortunately, evaluating the response of these functions is computationally intensive. The novel techniques hence propose to rethink HPO as a few-shot learning problem in which a shared deep surrogate model is trained to quickly adapt (with few response evaluations) to the response function of a new task.

## 3.2. Efficient deep learning using model compression and acceleration

Contributing partners: UNITN
As part of the activities related to this task, UNITN has participated in two important initiatives:

**Organization of workshop on Neural Architecture Search.** We organized a workshop[1] in conjunction with the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR) 2021. One of its goals was to bring together emerging research in the areas of automatic architecture search, optimization, hyperparameter optimization, data augmentation, representation learning and computer vision to discuss open challenges and opportunities ahead. Another goal was to benchmark lightweight NAS in a systematic and realistic approach. In this regard, we made a step forward in advancing existing state-of-the-art by organizing the first challenge and providing comprehensive benchmarks for fair comparisons. We set up three competition tracks, i.e., (1)

[1] https://cvpr21-nas.com

SuperNet[2], (2) Performance Prediction[3], and (3) Unseen Data[4], and we encouraged participants to propose novel solutions to advance the state-of-the-art. The challenge represented the first thorough quantitative evaluation on the topic of lightweight NAS. The workshop had 8 prominent invited speakers, i.e., Alan Yuille, Rongrong Ji, Gao Huang, Yi Ma, Kristen Grauman, Peter Vajda, Sara Sabour, and Chang Xu and attracted more than 100 participants.

**Organization of special issue on Deep Learning for Efficient and Robust Pattern Recognition.** We organized this issue in the Pattern Recognition journal[5] and, although its scope goes beyond NAS, one of the three main areas crucial to its subject, namely model compression and acceleration, is directly related to NAS. The special issue is going to be published in the December 2021 issue of the journal and will contain 30 accepted papers out of a total of 118 submitted manuscripts (an impressive number for a special issue!). For the special issue, we provided a survey paper covering the most important aspects of the pertinent research in the state of the art. Below, we detail the aspects directly related to NAS, i.e., model compression and acceleration.

### 3.2.1. Model Compression and Acceleration

Deep neural networks that achieve high precision in real-world AI tasks often come at the expense of many parameters, which in turn require significant computational resources and training time. Thus, there is a demand for model compression and acceleration techniques that would allow deployment to resource-constrained devices and real-time applications. In recent years, a growing number of approaches aim to compress and accelerate networks while making minimal compromises in terms of accuracy. Most of these can be classified into one of the following categories: parameter pruning, network quantization, low-rank factorization, model distillation, and compact network design.

**Parameter pruning.** Pruning is a popular technique that compresses networks by removing parameters with small impact to performance. Effectively, this reduces network complexity and avoids overfitting due to over-parameterization [1]. A typical pruning approach can be framed as an iterative process that repeats three stages: (1) evaluating parameter instances, (2) pruning parameters according to their importance, (3) fine-tuning to recover accuracy. Pruning can be either unstructured or be performed on any architectural granularities (e.g., neurons, weights, filters, channels, layers).

Non-structured pruning does not restrict the specific position of the parameters to be pruned and can thus lead to irregular structures. LeCun et al. [1] first eliminated unimportant weights from pre-trained networks and Han et al. [2] proposed an iterative method that removes weights whose magnitudes are below the predefined threshold and retrain the network to improve accuracy. This practice was later extended as a pipeline, namely deep compression [3], consisting of pruning, trained quantization, and Huffman coding. This pipeline achieves up to 35x reduction without sacrificing accuracy. However, it is impossible to recover mistakenly pruned parameters. Guo et al. [4] proposed dynamic network surgery to recover incorrectly pruned critical connections by adding a connection splicing operation after pruning. Another problem was that preset thresholds were not necessarily optimal and Li et al. [5] transformed threshold tuning into a constrained optimization problem, thus introducing more flexibility to the approach. Finally, NeST [6] combined gradient-

[2]https://aistudio.baidu.com/aistudio/competition/detail/64/0/introduction
[3]https://aistudio.baidu.com/aistudio/competition/detail/71/0/introduction
[4]https://competitions.codalab.org/competitions/29853
[5]https://www.sciencedirect.com/journal/pattern-recognition/special-issue/10N494KXDP8

based network growth and magnitude-based pruning to generate accurate and compact Deep Neural Networks (DNNs).

Non-structured pruning results in irregular sparse structures that may not fit well in most sparse computation libraries with parallel computation, which impedes actual compression performance. Also, non-structured pruning requires an additional index overhead. This has motivated the introduction of structured pruning approaches that better save computational resources in practice. Structured pruning can be applied on different granularities of structured sparsity. Anwar et al. [7] proposed an intra-kernel stride pruning method at kernel level for the first time. As for filter pruning, once a filter is pruned, both the corresponding channel in the next layer and the kernel accounting for the channel are also pruned. Thus filter pruning can significantly reduce the computation cost and memory overhead. Group-wise magnitude comparison [8] and Average Percentage of Zeros (APoZ) of channels [9] have been applied to filter pruning. Similar to dynamic network surgery [4], soft filter pruning [10] can also be used to adapt a dynamic pattern and thus improve model capabilities by recovering critical filters.

While the heuristic approaches described above can be effective, optimization methods are often preferred, as they exhibit improved model accuracy and better compression ratio. Structured sparsity learning [11] applied a LASSO regularization on structured weight groups, such as channels, filters, or layers. ThiNet [12] pruned filters based on statistical information from its next layer. The pruning problem was established as an optimization that minimized the reconstruction error by weighing the channels. Liu et al. [13] proposed to use the scaling factor in Batch Normalization to indicate the importance of channels. An L1-norm regularization was added onto the scaling factor, and channels with small scaling factors would be pruned. Ye et al. [14] proposed a different regularization to constrain the scaling factor and modify the bias to compensate for the error caused by pruning. The saliency map corresponding to each layer can also be regarded as an importance score for pruning [15].

Beyond research on pruning methods, some works have investigated whether parameter pruning is as valuable as we think. Frankle et al. [16] claimed that a randomly initialized dense network contains a small subnetwork that, when trained in isolation, can compete with the original. Thus, a pruned network can perform well when its weights are reset to their initial values and retrained. On the other hand, Liu et al. [17] claimed that only the pruned structure matters and not the initial weights; hence, fine-tuning pruned networks performs comparably or worse than training that model from scratch. As a result, rethinking the value of pruning would be an interesting topic.

**Network quantization.** Network quantization compresses the original network by reducing the number of bits required for parameter representation, thus reducing storage and computation requirements. Methods for network quantization can be roughly categorized into (a) storage compression and (b) acceleration.

Quantization methods for storage compression typically employ scalar or vector quantization techniques to reduce space requirements for storing network parameters. The original parameters are approximately represented by a codebook, consisting of a group of clustering centers and quantized codes indicating a mapping from original parameters to centers. With scalar or vector quantization, a high compression ratio can be reached. Gong et al. [18] applied scalar quantization to the weights of multilayer perceptron networks with k-means clustering and applied vector quantization to the fully connected layers with product quantization. Quantized CNN [19] utilized product quantization to compress both fully connected and convolutional layers by minimizing each layer‘s response error.

Most quantization techniques focus on accelerating computations. This is achieved by reducing the number of bits capturing neural activations or weight representations, where the common 32-

bit floating-point number representations are replaced by low-bit fixed-point ones that run faster. Some methods only quantize the weights of networks. For example, Gupta et al. [20] argued that 16-bit fixed-point numbers suffice to represent the weights of networks without sacrificing accuracy. For faster inference, many extremely low-bit quantizations are investigated. These include binary weight networks represent weights by only 1 bit, i.e., +1 or -1, and ternary weight networks represent weights by three values, i.e., +1, 0, or -1. BinaryConnect [21] implemented the training of the DNN with binary weights and obtained more than 32× compression. Binary Weight Network (BWN) [22] extended BinaryConnect by incorporating a scaling factor with binary weights to approximate original weights better. BWN achieved good performance on large-scale datasets such as ImageNet for the first time. BWNH [23] utilized hashing for training binary weight networks with the proposed alternating optimization method to learn the hash codes. Similar to BWN, Ternary Weight Network (TWN) [24] used ternary values that can be presented by 2 bits, which was more effective than their binary counterpart. Trained Ternary Quantization (TTQ) [25] simultaneously learned ternary weight values and scaling factors. There are also quantization approaches performed on both weights and activations. The bitwise neural network [26] introduced binary representations to weights, biases, activations and outputs, thus dramatically saving computation resources. Binarized Neural Networks (BNNs) [27], an extension to BinaryConnect, binarized both weights and activations and achieved comparable performance to the baseline on CIFAR-10. XNOR-Net [22] was proposed as an extension to BWN by binarizing both the weights and activations, achieving higher precision on ImageNet than BNN.

**Low-rank factorization.** A typical convolution layer has a filter that can be regarded as a 4D tensor, whose dimensions correspond to the output channel, input channel, height and width. Low-rank factorization techniques can be used to compress and accelerate networks by finding a low-rank tensor that is close to the original filter and can be easily decomposed for significantly reduced computation. Low-rank factorization methods can be classified as two-component, three-component, or four-component factorization, according to the number of factorized components. Two-component methods, such as Singular Value Decomposition (SVD), split a convolutional filter into two successive convolution layers. Jaderberg et al. [28] separated $k \times k$ filters into $k \times 1$ and $1 \times k$ filters. Zhang et al. [29] split filters with $d$ kernels into the succession of a filter with $d'$ kernels (the rank $d' < d$) of the same size and a filter consisting of $d$ $d' \times 1 \times 1$ kernels. A response reconstruction method was used to ensure the low-rank constraint and deal with nonlinear responses. Three-component methods factorize the convolutional filters into three low-rank tensors, i.e., three successive convolutional layers. For example, Zhang et al. [29] further used the factorization in Jaderberg et al. [28] to split the former filters, resulting in a three-component factorization that achieved more speed-up. One-shot whole network compression was presented in Kim et al. [30] based on the Tucker decomposition of kernel tensors. Specifically, the succession of filters with size $1 \times 1$, $k \times k$, and $1 \times 1$ was utilized to approximate the original filter. Four-component factorization considers the low-rank property on all four dimensions, i.e., input channel, output channel, height, and width. The Canonical Polyadic (CP) decomposition, an extension of SVD, was used in Lebedev et al. [31] to split the convolutional filter into four convolutions of size $1 \times 1$, $k \times 1$, $1 \times k$ and $1 \times 1$. However, CP decomposition would result in a relatively large approximation error.

**Model distillation.** Unlike the above approaches, model distillation uses knowledge transferred from a large trained network (teacher network) to train a smaller network (student network) that reaches comparable accuracy. Usually, the transfer knowledge is learned by mimicking the class distributions output via softmax. The knowledge distillation method in [32] followed the idea of learning from the teacher network's output softmax distribution, where the high temperature was placed into softmax to soften the teacher's output during training. FitNets [33] was a framework proposed to train a thin but deep student network. With a deeper architecture, the student network can achieve higher accuracy. The student network also took advantage of intermediate representations of the teacher as distilled knowledge during training. Attention mechanisms can provide better intermediate representations, so distilling knowledge by mimicking the attention maps of the teacher networks was effective to enhance the performance of the student network [34]. Distillation methods have also been used in uncertainty quantification, which transfers knowledge from multiple outputs acquired from Monte-Carlo (MC) dropout or deep ensembles, thus making epistemic uncertainty estimation in a single pass possible [35, 36].

**Compact model design.** Besides approaches that rely on pre-trained deep architectures, there are also attempts to design compact architectures aiming at low cost and fast inference, which is suitable for deployment on mobile devices and real-time applications.

Some compact architectures focused on enhancing the spatial correlation by enlarging the reception field of the convolutional filters efficiently, such as dilated convolutions [37] and deformable convolutions [38]. The majority of compact model design approaches focused on enhancing the channel correlations within a conv layer or between conv layers. The most famous ResNet [39] used residual connections for inter-layer correlation, and later DenseNet [40] used dense connections between layers, yielding a more condensed model. More compression can be obtained by adopting the topology of channels within a conv layer. The convolution was thus divided into a convolution for each channel to capture the spatial correlation and linear aggregation of outputs from all channels to capture the channel correlation (usually as a point-wise convolution [41]). The bottleneck architecture was later proposed using two point-wise convolutions [42]. The former one projected the feature map to a low-dimensional feature space, and the latter reprojected the feature map, which passed through a standard convolution, back to the original feature space. SqueezeNet [43] used a point-wise convolution for feature map squeezing and used two branches, one with $1 \times 1$ conv and the other with $3 \times 3$, for feature map expansion. Group convolution is another commonly used method. The feature map is split into several groups. Each group passes through standard convolutions and then concatenates back. ResNeXt [44] was a variant of ResNet with group convolution. ShuffleNet [45] added a channel shuffle operation to the point-wise group convolution in ResBlock, enhancing the expressive power of the network, and ShuffleNetV2 [46] derived several practical guidelines for further improvement. An extreme case of group convolution is depth-wise convolution, where each channel forms a group. Depth-wise separable convolutions, introduced in MobileNet [47], consisted of a depth-wise convolution and a point-wise convolution, where the spatial and channel correlations were separately treated. MobileNetV2 [48] introduced an inverted residual block, which expanded input channels before depth-wise convolution and then squeezed it back. Such a module significantly improved the network's performance with little overhead increase since depth-wise convolution was lightweight.

Overall from this comprehensive analysis of the representative works and recent developments in efficient and robust deep learning (including NAS) one can conclude that the main trends are in (1) improving the interpretability of deep learning methods, (2) designing compact and efficient network architectures in specific pattern recognition problems, (3) designing novel adversarial attacks and (4) investigating training stability. Recent interest in explainable artificial intelligence (covered mostly in WP4), especially from governments within the European General Data Protection Regulation (GDPR), shows the vital consideration of AI's ethics, trust, and bias. This is also covered in WP2.

### 3.2.2. Relevant publications

- *X. Bai, X. Liu, Q. Liu, J. Song, N. Sebe and B. Kim*, Explainable Deep Learning for Efficient and Robust Pattern Recognition: A Survey of Recent Developments, *Pattern Recognition, vol. 120, Article 108102, December 2021* [49]
Zenodo record: https://zenodo.org/record/5442806.

## 3.3. Deep kernel surrogates for few-shot Bayesian hyperparameter optimization

**Contributing partners:** IBM

### 3.3.1. Problem Statement

NAS aims to automate the optimisation of machine learning neural architectures for given AI tasks. However, many machine learning models have very sensitive hyperparameters that must be carefully tuned for efficient use. Unfortunately, finding the right setting is a tedious trial and error process that requires expert knowledge. State-of-the-art methods address this problem by providing tools to automate hyperparameter optimization, where Bayesian optimization has become the standard for this task [50]. It treats the problem of hyperparameter optimization as a black box optimization problem. Here the black box function is the hyperparameter response function, which maps a hyperparameter setting to the validation loss. Bayesian optimization consists of two parts. First, a surrogate model, often a Gaussian Process (GP), is used to approximate the response function. Second, an acquisition function is used that balances the trade-off between exploration and exploitation. In a sequential process, hyperparameter settings are selected and evaluated, followed by an update of the surrogate model.

Unfortunately, evaluating the response function is computationally intensive. As a remedy, earlier work emphasizes the need for transfer learning surrogates which learn to optimize hyperparameters for an algorithm from other tasks. Recently, several attempts have been made to extend Bayesian optimization to account for a transfer learning setup. It is assumed here that historical information on machine learning algorithms is available with different hyperparameters. This can either be because this information is publicly available (e.g. OpenML[6]) or because the algorithm is repeatedly optimized for different data sets. To this end, several transfer learning surrogates have been proposed that use this additional information to reduce the convergence time of Bayesian optimization.

### 3.3.2. Methodology

In contrast to previous work, we propose to rethink Hyperparameter Optimisation (HPO) as a few-shot learning problem in which we train a shared deep surrogate model to quickly adapt (with few response evaluations) to the response function of a new task. We propose the use of a deep kernel network for a GP surrogate that is meta-learned in an end-to-end fashion in order to jointly approximate the response functions of a collection of training data sets.

We propose a new paradigm for accomplishing the knowledge transfer by reconceptualizing the process as a few-shot learning task. Inspiration is drawn from the fact that there are a limited number of black box function evaluations for a new HPO task (i.e. few shots) but there are ample evaluations of related black box objectives (i.e. evaluated hyperparameters on other data sets). This approach has several advantages. First, a single model is learned that is trained to quickly

[6] https://www.openml.org

adapt to a new task when few examples are available. This is exactly the challenge we face when optimizing hyperparameters. Second, this method can scale very well to any number of considered tasks. This not only enables the learning from large metadata sets but also enables the problem of label normalization to be dealt with in a new way.

Given $T$ related source tasks and very few examples of the target task, we want to make reliable predictions. For each of the source tasks we have observations $D^{(t)} = \{(x_i^{(t)}, y_i^{(t)})\}_{i=1...n(t)}$ , where $x_i^{(t)}$ is a hyperparameter setting and $y_i^{(t)}$ is the noisy observations of $f^{(t)}(x_i^{(t)})$, i.e. a validation score of a machine learning model on data set $t$. In the following, we will denote the set of all data points by $D := \cup_{t=1}^{T} D^{(t)}$. We propose to use an adaptation for GPs [51] as a surrogate model within the Bayesian optimization framework.

A deep kernel $\varphi$ is used to learn parameters across tasks such that all its parameters $\theta$ and $\omega$ are task-independent. All task-dependent parameters are kept separate, which allows to marginalize its corresponding variable out when solely optimizing for the task-independent parameters. If we assume that the posteriors over $\theta$ and $\omega$ are dominated by their respective maximum likelihood estimates $\theta^{'}$ and $\omega^{'}$ , we can approximate the posterior predictive distribution by

$$p(f_*|x_*, D) = \int p(f_*|x_*, \theta, \omega)p(\theta, \omega|D)d\theta, \omega \approx p(f_*|x_*, D, \theta^{'}, \omega^{'}) \tag{1}$$

$$\log p(y^{(1)}, ..., y^{(T)}|X^{(1)}, ..., X^{(T)}, \theta, \omega) = \sum_{t=1}^{T} \log p(y^{(t)}|X^{(t)}, \theta, \omega) \tag{2}$$

$$\alpha - \sum_{t=1}^{T} ({y^{(t)}}^T {K_n^{(t)}}^{-1} y^{(t)} + \log |K_n^{(t)}|) \tag{3}$$

We estimate $\theta^{'}$ and $\omega^{'}$ by maximizing the log marginal likelihood for all tasks. We maximize the marginal likelihood with Stochastic Gradient Ascent (SGA). Each batch contains data for one task. It is important to note that this batch data does not correspond to the data in $D^{(t)}$, but rather a subset of it. The training works as follows. In each iteration, a task $t$ is sampled uniformly at random from all $T$ tasks. Then we sample a batch of training instances uniformly at random from $D^{(t)}$. Finally, we calculate the log marginal likelihood for this batch (Equation 2) and update the kernel parameters with one step in the direction of the gradient.

In HPO, we are only interested in the hyperparameter setting that works best according to a predefined metric of interest. Therefore, only the ranking of the hyperparameter settings is important, not the actual value of the metric. Therefore, we are interested in a surrogate model whose prediction strongly correlates with the response function and the squared error is of little to no interest for us. In practice, however, the minimum / maximum score and the range of values differ significantly between the tasks, which makes it challenging to obtain a strongly correlating surrogate model. The most common way to address this problem is to normalize the labels.

We propose a task augmentation strategy that addresses the label normalization issue by randomly scaling the labels for each batch. Since $y_{min}$ and $y_{max}$ are the minimum and maximum values for all $T$ tasks, we can generate augmented tasks for each batch $B = (x_i, y_i)_{i=1,...,b} \sim D^{(t)}$, where $b$ is the batch size, as follows. A lower and upper limit is sampled for each sample batch, such that $l < u$ holds. Then the labels for that batch are scaled to this range,

$$l \sim \cup(y_{min}, y_{max}), u \sim \cup(y_{min}, y_{max}) \tag{4}$$

$$y \leftarrow \frac{y - l}{u - l} \tag{5}$$

**Algorithm 1** Few-Shot GP Surrogate

```
Require: Learning Rate $\alpha$ and $\beta$, training data $D$, kernel $k$ and neural network $\varphi$
  while model invariance not achieved do
    Sample task $t \sim \cup(1, .....T)$;
    Estimate $l$ and $u$ (Equation 4)
    for $b_n$ times do
      Sample batch $B = ((x_i, y_i))_{i=1,...,b} \sim D^{(t)}$ and scale labels y using $l$ and $u$;
      Compute marginal likelihood $L$ on $B$. (Equation 2);
      $\theta \leftarrow \theta + \alpha\nabla_\theta L$;
      $\omega \leftarrow \omega + \beta\nabla_\omega L$;
```

| Method | AdaBoost | | | GLMNet | | | SVM | | |
|---|---|---|---|---|---|---|---|---|---|
| | 15 | 33 | 50 | 15 | 33 | 50 | 15 | 33 | 50 |
| Random | 4.87 | 3.02 | 2.16 | 0.85 | 0.40 | 0.29 | 2.01 | 1.12 | 0.83 |
| GP (LHS) | 3.87 | 2.49 | 2.23 | 1.32 | 1.01 | 0.73 | 1.94 | 1.40 | 1.14 |
| GP (WS) | 3.25 | 1.66 | 1.02 | 0.86 | 0.36 | 0.24 | 1.10 | 0.81 | 0.65 |
| RGPE | 5.29 | 3.26 | 2.83 | - | - | - | - | - | - |
| MetaBO | 5.27 | 3.52 | 1.96 | 11.02 | 10.97 | 10.96 | 12.39 | 12.39 | 11.93 |
| ABLR | 4.56 | 1.44 | 1.24 | 1.77 | 0.50 | 0.40 | 1.81 | 1.23 | 0.84 |
| ABLR (WS) | 3.17 | 1.72 | 1.17 | 0.54 | 0.36 | 0.29 | 1.47 | 1.08 | 0.87 |
| Multi-Head GPS (WS) | 6.78 | 3.45 | 1.80 | 2.83 | 2.80 | 2.68 | 11.20 | 9.82 | 8.72 |
| FSBO | **3.10** | **1.13** | **0.80** | **0.42** | **0.22** | **0.16** | **0.79** | **0.51** | **0.36** |

*Table 1. FSBO obtains better results for all hyperparameter optimization problems. The best results are in bold. Results that are not significantly worse than the best are in italics. Used initialization in parentheses: Latin Hypercube Sampling (LHS) and Warm Start (WS).*

### 3.3.3. Experiments

We used three different optimization algorithms to compare the different hyperparameter optimization methods: AdaBoost, GLMNet, and SVM. We created the GLMNet and SVM metadata set by downloading the 30 data sets with the most reported hyperparameter settings from OpenML for each problem. The AdaBoost data set is publicly available [52]. The number of settings per data set varies, the number of settings across all tasks for the GLMNet problem is approximately 800,000. We compare the following list of hyperparameter optimization methods. Random Search [53], GP [50], RGPE [54], MetaBO [55], ABLR [56], Multi-Head GPs and our proposed method Few-Shot Bayesian Optimization (FSBO).

The experiments are repeated 10 times and evaluated in a leave-one-task-out cross-validation. This means that all transfer learning methods use one task as the target task and all other tasks as source tasks. For AdaBoost we use the same train/test split as used by [55] instead. We report the aggregated results for all tasks within one problem class with respect to the mean of normalized regrets. Regret is a standard common metric used within this field which measures the difference between the reward of a possible action and the reward of the action actually taken. The normalized regret for a task is obtained by first scaling the response function values between 0 and 1 before calculating the regret.

We conduct experiments on three different metadata sets and report the aggregated mean of normalized regrets in Table 1. The best results are in bold. Results that are not significantly worse than the best are in italics. We determined the significance using the Wilcoxon signed rank test with a confidence level of 95%. Results are reported after every 33 trials of Bayesian optimization for GLMNet and SVM. Since AdaBoost has fewer test examples, we report its results in shorter intervals.

Our proposed FSBO method outperforms all other transfer methods in all three tasks. Results are significantly better except in the case of AdaBoost where GP(WS) achieves similar but on average worse results. The results obtained for MetaBO are the worst. The problem is apparently that the Reinforcement Learning method does not work well for larger number of trials. Also, ABLR and the very related Multi-Head GP are not performing very well. One possible reason for this might be because the neural network parameters are fixed which prohibits a possibly required adaptation to the target task. The vanilla GP and its transfer variant that uses a warm start turn out to be among the strongest baselines. This is in particular true for the warm start version which is the second best method. This simple baseline of knowledge transfer is often not taken into account when comparing transfer surrogates, although it is easy to implement. Due to the very competitive results, we recommend using it as a standard baseline to assess the usefulness of new transfer surrogates.

Figure 1 highlights the components that make significant differences to ABLR. Multi-Head GP is a surrogate model that shares the neural network between tasks but uses a separate GP for each task. It closely resembles the idea of ABLR but is closer to our proposed implementation of FSBO. Starting from this configuration, we add components that eventually lead to our proposed model FSBO. We consider Multi-Head GP (WS), a version that additionally uses the warm start method instead of a random initialization. Multi-Head GP (fine-tune) not only updates the kernel parameters when a new observation is received but also fine-tunes the parameters of the neural network. Finally, FSBO is our proposed method, which uses only one GP for all tasks. We see that all Multi-Head GP versions have trouble adapting to the target tasks efficiently. Fine-tuning the deep kernel is an important part of learning FSBO. Although FSBO outperforms all Multi-Head GP versions without fine-tuning, the additional use of the latter induces a significant improvement. We analyzed the reason for this in more detail. In terms of the standard "exploitation vs exploration" search algorithm objectives, we observed that the learned neural network provides a strong prior that leads to strong exploitation behavior. In other words, the neural network tends to produce

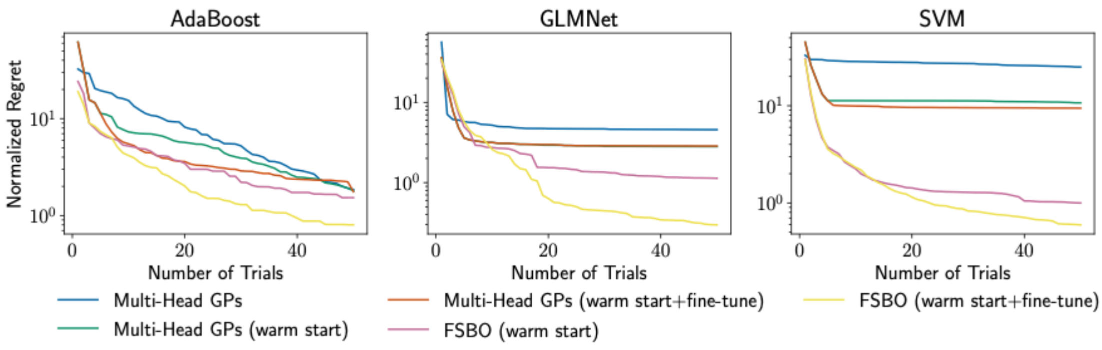


*Figure 1. Comparison of the contribution of the various FSBO components to the final solution. Each component makes its own contribution.*

similar architectures instead of exploring yet unexplored alternatives. Fine-tuning prevents this and thus does not get stuck in local optima.

In this work, we proposed to rethink hyperparameter optimization as a few-shot learning problem in which we train a shared deep surrogate model to quickly adapt (with few response evaluations) to the response function of a new task. We propose the use of a deep kernel network for a GP surrogate that is meta-learned in an end-to-end fashion in order to jointly approximate the response functions of a collection of training data sets. By doing so, we set the new state of the art in transfer learning for the HPO and provide ample evidence that we outperform strong baselines published in ICLR and NeurIPS by a statistically significant margin.

### 3.3.4. Relevant publications

- *Martin Wistuba, Josif Grabocka*, Few-Shot Bayesian Optimization with Deep Kernel Surrogates, *Ninth International Conference on Learning Representations ICLR, 2021* [57]
  Zenodo record: https://zenodo.org/record/4556929

# 4. AI at the Edge, Decentralised and Distributed Learning (Task 3.5)

## 4.1. Overview of our AI at the edge, decentralised and distributed learning contributions

The pervasive integration of mobile devices and the internet-of-things in every aspect of people's lives has created new kinds of media data (e.g., social network interactions, multimedia features, sensor readings) at unprecedented scales that reach big data and beyond [58, 59, 60, 61]. Traditional AI training and deployment paradigms consider centralized (web) services that can be queried through one endpoint. However, services with computationally heavy AI struggle to gather and process the ever-expanding resources of the interconnected world [62, 63, 64, 65]. Even worse, real world data rarely conform to data homogeneity (e.g., identically distributed data) often assumed by machine learning model designers. These concerns have motivated the creation of new machine learning paradigms that can cover heterogeneous global-scale data. One emerging direction, which we follow in this task, is to share the burden of AI computations across many devices. In addition to addressing scalability, this can also meet privacy and confidentiality requirements by either sharing obfuscated versions of local device data with others (e.g., derivatives with additive noise) [66, 67, 68] or not sharing any data at all. We address three ways of splitting computations across multiple devices.

First, we take advantage of the total processing power of "edge" devices gathering data, such as smart phones and sensors. Given that these maintain a base minimum of processing power, accumulated computational capabilities grow proportionally to the total volume of generated data. Thus, we propose that devices can collectively learn approximate machine learning models without the need for central supervision by letting each device hold its own approximation of ideal models and employing communication exchange schemes to collectively learn across models.

Second, we investigate promising methods that can be used to increase the efficiency of distributed training over clusters of high-performance workstations (usually equipped with multiple GPUs). To this end, we consider improvements of centralized computational models that are applicable on data batches and are thus easy to compute separately on each workstation once they are ported in a distributed training setting.

Third, we deploy pre-trained models that perform inference by processing local data inside devices—a task that would otherwise be delegated to central services. In this direction, we tackle inference under changing resource constraints. In particular, edge devices provide limited computational capabilities and energy [45, 69, 70, 71]. However, these constraints may change over time, for example due to devices moving around (bandwidth changes) or when other processes start or stop in shared computing environments. Adapting to such changes at runtime is critical for the deployment of stable AI tools [72, 73, 74, 75, 76, 77].

Current work in T3.5 pursues the above-mentioned paradigms of offloading AI computations to edge devices and on strategies for improving model generalization.

**CERTH** analysed training of AI models when edge devices are organized into unstructured peer-to-peer communication graphs, where communications occur only between linked devices, new links are either not formed or slowly created over time and devices are linked and communicate with each other based on underlying relations, such as friendship or proximity. Thus, social network overlays coincide with communication networks (e.g., social network friend messages coincide with communication between their linked devices). Two highly interconnected AI practices were investigated to take advantage of link information: random walks across social network overlays that smooth information with a stochastic equivalent of exact random walks (Subsection 4.2.2) and graph neural networks that leverage links to improve the accuracy of predictive classifiers (Subsection 4.2.3). These run on top of existing decentralized peer-to-peer network links under uncertain link availability to perform few-shot node recommendation and classify nodes that hold few (e.g., one) data points each.

**JR** experimented with novel strategies (mini-batch trimming and random sample dropping) aimed at increasing the *generalization capability* of trained neural network models (Subsection 4.3). The focus was on developing methods which are minimally invasive (e.g. do not make it necessary to modify the network structure itself) and which could later be ported efficiently to distributed training frameworks. Evaluations with different models on various image classification datasets show that mini-batch trimming improves the model generalization, whereas random sample dropping does not lead to an improvement.

**IDF** developed a flexible Recurrent Neural Network (RNN) that adapts to changes in computational resources by smartly limiting processed inputs (Subsection 4.4.1), as well as an improved training procedure for flexible models based on the idea of a *teacher assistant* (Subsection 4.4.2).

## 4.2. Decentralized information diffusion and graph neural networks

**Contributing partners:** CERTH

### 4.2.1. Problem setting

We tackle the problem of classifying points of a shared feature space when each one is stored at the device generating it, i.e. each device accesses only its own point but all devices collect the same features. For example, mobile devices of decentralized social media users could classify user interests based on locally stored content features, such as the bag-of-words of sent messages, and given that some device users have disclosed their interests. We further consider devices that are nodes of peer-to-peer networks and communicate with each other based on underlying relations, such as friendship or proximity. In this setting, social network overlays coincide with communication networks. At the same time, social behavior dynamics (e.g. users going online or offline) could prevent devices from communicating on-demand or at regular intervals. Ultimately, with who and when communication takes place can not be controlled by learning algorithms. Finally, enforcing random parameter exchanges between devices that have not established real-world connections, such as direct messages and user-driven status updates, introduces both routing overheads and privacy concerns. In fact, these shortcomings have motivated the introduction of peer-to-peer social network-based exchange schemes, in which user devices directly communicate only with small subsets of linked ones.

**Decentralized learning.** Various schemes have been proposed for learning in environments without a central conductor, most falling under the umbrella of *gossip protocols* that randomly exchange and aggregate parameters (e.g., via averaging) between pairs of decentralized devices [78, 79, 80, 81, 82].

**Graph diffusion.** Graph diffusion refers to smoothing node values (scores or representations) across graphs by performing a weighted smoothing of other values, i.e., a type of weighted averaging that depends on edge importance and how many edge hops away other nodes reside. Popular approaches involve exponentially decreasing the importance of nodes more hops away using among others Markovian random walks with restart [83, 84] and heat kernel weights [85]. Modern understanding of this setting has evolved into graph filtering [86] that performs a node domain equivalent of graph spectral transformations.

**Graph Neural Networks (GNNs).** GNNs are a machine learning paradigm in which links between data samples[7] are used to improve the predictions of base neural network models [87]. Samples are linked to form graphs based on real-world relations, and information diffusion schemes propagate their latent attributes to neighbors. Then, they are aggregated – for example, via averaging – and transformed with dense layers and non-linear activations to new representations to be propagated again. This is repeated either ad infinitum or for a fixed number of steps to combine original representations with structural information. Although propagation is similar to decentralized learning in that nodes work independently, transformation parameters are shared and learned across all nodes. It has been argued that the success of GNNs can largely be attributed to the use of recurrence [88, 89] rather than end-to-end training of seamless architectures. As a result, recent works have introduced decoupled architectures that achieve the same theoretical expressive power as end-to-end training by training base statistical models (such as two-layer perceptrons) to make predictions, and smoothing the latter through graph edges [90, 91].

**Uncertain peer-to-peer communication.** To provide a framework in which peer-to-peer nodes learn to classify themselves, we specify a communication protocol of information exchanges. First, we consider static matrices $A$ with elements $A[u,v] = \{1 \text{ if } (u,v) \text{ are linked}, 0 \text{ otherwise}\}$, where links indicate communication channels of uncertain availability. These matrices are not fully observable by decentralized nodes (nodes are at most aware of their corresponding rows and columns), but would be the input to equivalent centralized GNNs. Communication uncertainty is encoded using time-evolving communication matrices $A_{com}(t)$ with non-zero elements indicating exchanges through the corresponding links:

$$A_{com}(t)[u,v] = \{1 \text{ if } u,v \text{ linked and communicate at time } t, 0 \text{ otherwise}\}$$

To simplify the our analysis, and without loss of generality, we adopt a discrete notion of time $t = 0, 1, 2, \dots$ that orders the sequence of communication events.

To exchange information through channels represented by time-evolving communication matrices, we use the broadly popular *send-receive-acknowledge* communication protocol: devices $u$ (in our case, these are nodes of the peer-to-peer network) are equipped with identifiers $u.id$ and operations $u.$SEND, $u.$RECEIVE and $u.$ACKNOWLEDGE that respectively implement message generation, receiving message callbacks that generate new message to send back and acknowledging that sent messages have been received. Usage of these operations is demonstrated in Algorithm 2.

**Algorithm 2** Send-Receive-Acknowledge protocol

```
Inputs: devices $u \in V$ with identifiers $u.id$, time-evolving $A_{com}(t) : V \times V \to \mathbb{R}$
for $t = 0, 1, 2, \dots$ do
    for all $(u,v)$ such that $A_{com}(t)[u,v] = 1$ do
        message← $u.$SEND(v.id)
        reply← $v.$RECEIVE(u.id, message)
        $u.$ACKNOWLEDGE($v.$id, reply)
```

[7]In our setting there is 1-1 correspondence between samples and graph nodes.

### 4.2.2. Uncertain peer-to-peer random walk diffusion

Information diffusion is a popular graph analysis method, in which prior scores of nodes are propagated (e.g. averaged) through graph neighbors. Our analysis tackles the following generalized form, that obtains node values $\pi_\infty$ in the form of vectors in a Hilbert space, where $\pi_\infty[u]$ correspond to eventual values of nodes $u$, by accounting for different types of matrix normalization and ways to bias towards initial node values $\pi_0$:

$$\pi_\infty = (\mathcal{I} - aD^{-d}AD^{d-1})^{-1}P_\gamma\pi_0, \tag{6}$$

where $\mathcal{I}$ is the unit matrix, $P_\gamma = diag(\{1-a \text{ if } u \in V_{train}, 1-\gamma \text{ otherwise}\}_u)$ is a diagonal matrix that scales differently non-training and training node personalization, $D = diag([\sum_v A[u,v]]_u)$ is a diagonal matrix of node degrees and $a \in [0,1]$ is a diffusion parameter. For $\gamma = a$, this is equivalent to constraining the personalized PageRank scheme [84] with normalized communication matrices $D^{-d}AD^{d-1}$ that preserve original node representations $\pi_n[v] = \pi_0[v]$ assigned to training nodes $v \in V_{train}$. This closed form is equivalent to restoring training node scores after each power method iteration $\pi_{n+1} = aD^{-p}AD^{1-p}\pi_n + (1-a)\pi_0$, where each iteration step is a specific type of graph convolution. On the other hand, for $\gamma = 1$ only the personalization $\pi_0[v]$ of training nodes $v$ is diffused in the graph.

To set up a decentralized implementation of personalized PageRank, we introduce a theoretical construct we dub *decentralized graph signals*. This describes decentralized operations in peer-to-peer networks while accounting for personalization updates over time, in case these are trained while being diffused. Our structure is defined as matrices $S : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^F$ with multidimensional vector elements $S[u,v] \in \mathbb{R}^F$ (in our case $F$ is the number of classes) that hold in devices $u$ the estimate of device $v$ representations. Rows $S[u]$ are stored on devices $u$ and only cross-column operations are impacted by communication constraints.

We consider a scheme that updates decentralized graph signals $S(t)$ at times $t$ per the rules:

$$\begin{aligned} S(t)[u,v] &= S(t-1)[u,v] + A_{com}(t)[u,v]\left(\tfrac{S(t-1)[v][v]}{D[u,u]^p} - S(t-1)[u,v]\right) \\ S(t)[u,u] &= P_\gamma[u,u]S_0(t)[u] + a\sum_v \tfrac{A_{masked}[u,v]}{D[u,u]^{1-p}} S(t)[v][v] \end{aligned} \tag{7}$$

where $S_0(t)[u] \in \mathbb{R}^F$ are time-evolving representations of nodes $u$. The first of the above equations describes node representation exchanges between devices based on the communication matrix, whereas the second one performs a local update of personalized PageRank estimation given the last updated neighbor estimation that involves only data stored on device $u$. In [92] we show that, if $\lim_{t\to\infty} S_0(t)[u] = \pi_0[u]$ is bounded and converges in distribution with linear rate then expected error between $\lim_{t\to\infty} S(t)[u,u]$ and $\pi_\infty[u]$ satisfying Equation 6 converges in distribution with linear rate, as long as there is a lower bound to the communication probability between devices, i.e., they never stop communicating completely.

### 4.2.3. Uncertain peer-to-peer decoupled GNNs

When network nodes corresponding to data points are linked based on social relations, a lot of information resides in their link structure in addition to data features. For instance, social network nodes one hop away often assume similar classification labels, a concept known as homophily [93, 94]. Yet, existing decentralized learning algorithms do not naturally account for information encapsulated in links, as they are designed for structured networks of artificially generated topologies. In particular, decentralized learning often focuses on creating custom communication topologies that optimize some aspect of learning [82] and are thus independent from data. Our investigation differs from this assumption in that we classify data points stored in decentralized devices that form unstructured communication networks. In these, links capture real-world activity (e.g. social interactions) unknown at algorithm design time and produce unstructured topologies.

Let us consider base classifiers $R_\theta(\cdot)$ with parameters $\theta$. These are trained on a set of labeled nodes $V_{train}$ to produce predictions $\hat{y} = R_\theta(x)$ from feature vectors $x$, where desired predictions one-hot encode classification labels. Regardless of how these are trained, peer-to-peer network links adhering to the principle of homophily, where similarly-attributed nodes tend to be connected to each other, can be leveraged to improve predictions. For neural networks, this is achieved by transforming them to GNNs that incorporate graph convolutions in multilayer parameter-based transformations both during training and during predictions.

Unfortunately, graph convolutions aggregate latent node neighbor representations but uncertain availability means that either the last retrieved representations should be used or node features and links from many hops away should be stored locally for in-device computation of graph convolutions. In the first case, convergence to equivalent centralized model parameters could be slow, since learning would impact neighbor representations only during communication. In the second case, multilayer architectures aiming to broaden node receptive fields from many hops away would end up storing most network links and node features in each node.

To avoid these shortcomings, we build on decoupled GNNs, which separate the challenges of training base classifiers with leveraging network links to improve predictions. To manipulate learning primitives, we organize base predictions of node features $X$, where rows $X[u]$ hold the features of nodes $u$, into matrices $R_\theta(X)$ with rows holding the predictions of respective feature rows $R_\theta(X)[u] = R_\theta(X[u])$. Predicted classes would be obtained by $\text{argmax} R_\theta(X)[u]$. If classifiers are trained for the features and labels of node sets $V_{train}$, we build on the FDiff-scale decoupled GNN's description [90], which we transcribe as:

$$\hat{Y} = (\mathcal{I} - \alpha D^{-0.5} A D^{-0.5})^{-1} P_\alpha \big(R_\theta(X) + (1-\alpha) s (\mathcal{I} - A_{masked} D^{-1})^{-1} P_1 (Y - R_\theta(X))\big) \tag{8}$$

where masked adjacency matrix elements $A_{masked}[u, v] = \{1 \text{ if } u = v \text{ and } u \in V_{train}, A[u, v] \text{ if } u \notin V_{train}, 0 \text{ otherwise}\}$ (these are not symmetric) prevent diffusion from affecting training nodes, $P_\gamma = \frac{1}{1-\alpha} diag(\{1 - \alpha \text{ if } u \in V_{train}, 1 - \gamma \text{ otherwise}\}_u)$ and $\alpha \in [0, 1]$, $s \in \mathbb{R}$ are hyperparameters

This scheme requires the training of a base predictor and the diffusion of predictions through the graph with Equation 8. The latter comprises two random walk diffusion steps adhering to the form of Equation 6 for parameters $(\gamma, d, \alpha)$ equal to $(\alpha, 0.5, \alpha)$ and $(1, 0, 1)$ respectively, applied on adjacency matrices $A$ and their modified versions $A_{masked}$ (with a respective communication matrix). Thus, decentralized graph signals can approximate numerical outputs per Equation 7. Notably, given that the base learning algorithm exhibits linear convergence in distribution, this property also transfers to the decentralized graph signal computations and hence the overall decentralized graph neural network also converges at a linear rate.

### 4.2.4. Experiments

To compare the ability of peer-to-peer learning algorithms to make accurate predictions, we experiment on three datasets that are often used to assess the quality of GNNs [95], a pre-processed version of which alongside common train-validation-test sets of nodes we retrieve using the programming interface of the publicly available Deep Graph Library [96]. These are the Citeseer [97], Cora [98] and Pubmed [97] social graphs for document classification based on textual features and relations between documents. For these, we simulate peer-to-peer networks with the same nodes and links as the datasets and fixed probabilities for edge communication at each time step by uniformly sampling them from the range $[0, 0.1]$ (similar results are obtained for communication rates sampled from different intervals). We repeat experiments 5 times and report classification accuracy on test labels after 1000 time steps (all algorithms converge well within that time).

Experiments employ three base classifiers: (a) *MLP* – a multilayer perceptron often employed by GNNs [99, 90]. This consists of a dense two-layer architecture starting from a transformation of node features into 64-dimensional representations activating ReLU outputs and an additional dense transformation of the latter whose softmax aims to predict an one-hot encoding of classification labels. (b) *LR* – a simple logistic regression classifier. (c) *Label* – classification that repeats training node labels. If no diffusion is performed, this provides random predictions for test nodes.

MLP and LR are trained towards minimizing a cross-entropy loss with an Adam optimizer [100, 101]. We set learning rate $0.1^L$, where $L$ is the number of dense layers, and maintain the default momentum parameters proposed by the optimizer's original publication. For MLP, we use 50% dropout for the dense layer to improve robustness and for all classifies we L2-regularize dense layer weights with 0.0005 penalty. We do not perform hyperparameter tuning, as further protocols would be needed to make peer-to-peer nodes learn a common architecture optimal for a set of validation nodes, for example by porting HPO of Subsection 3.3 to the decentralized setting. For FDiff-scale hyperparameters, we select a personalized PageRank restart probability often used for graphs with several thousand nodes $1 - \alpha = 0.1$ and equal prediction and error scale parameter $s = 1$, where the latter is selected so that it theoretically satisfies the heuristic of perfectly reconstructing the class labels of training nodes.

Table 2 compares the accuracy of base algorithms vs. their augmented predictions with the decentralized p2pGNN and a fully centralized implementation of FDiff-scale. The last two schemes implement the same architecture and differ only on whether they run on peer-to-peer networks or not respectively. Comparisons span pre-trained versions of classifiers and ones fully decentralized trained with gossip protocols, where the latter make use of both training and validation nodes, as it is not possible to use validation nodes for robust stopping point detection. In both cases, diffusion makes use of both training and validation nodes, as this does not affect base classifier training. Fully centralized GNNs always use the pre-trained base classifiers, since these are the centralized counterpeparts of gossip training.

Overall, experiment results indicate that, at least for pre-trained classifiers, p2pGNN successfully applies GNN principles to improve base classifier accuracy, whereas gossip classifiers tend to "leak" the graph structure to parameters and hence are sometimes similar to simple label diffusion. Mitigiating this last issue is a subject of our ongoing research. Importantly, although neighbor-based gossip training of base classifiers on both training and validation nodes outperforms models pre-trained on only training nodes (in which case validation nodes are used for early stopping), decentralized graph diffusion of the latter exhibits the highest accuracy across most combinations of datasets and base classifiers.

#### 4.2.5. JGNN: a library to support learning on edge devices

Concepts discussed throughout section 4.2, such as gossip learning, require the ability to perform AI computations on edge devices. Existing decentralized AI frameworks that are catered for this setting, such as TensorFlow lite [102], typically follow a pipeline where models are pre-trained in computationally-capable infrastructure and are only deployed to perform inference on devices without being able to learn there. This covers in large part the requirements of the *PretrainedDiff* model presented in the previous section (decentralized graph signals are easy to implement and perform lightweight operations during communication), but does not support settings where decentralized AI originally designed for GPU-savvy systems is trained within edge devices of restricted resources, for example to perform gossip learning. This prospect is particularly important in fully decentralized applications, in which pre-trained models can not be hosted in one place, for example

*Table 2. Comparing the accuracy of different types and training schemes of base algorithms and their combination with the diffusion of p2pGNN. Accuracy is computed after* 1000 *time steps and averaged across* 5 *peer-to-peer simulation runs.*

| | **Base** | | | **p2pGNN** | | | **Fully Centralized GNN** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Citeseer | Cora | Pubmed | Citeseer | Cora | Pubmed | Citeseer | Cora | Pubmed |
| | | | | | **Pre-trained** | | | | |
| MLP | 52.3% | 54.9% | 70.9% | 67.8% | 81.5% | 76.0% | 69.0% | 84.0% | 81.2% |
| LR | 59.4% | 58.7% | 72.2% | 70.5% | 82.0% | 77.3% | 70.3% | 85.7% | 81.5% |
| | | | | | **Gossip** | | | | |
| MLP | 63.1% | 66.3% | 74.9% | 61.3% | 80.8% | 78.0% | 69.0% | 84.0% | 81.2% |
| LR | 61.8% | 79.9% | 78.7% | 61.4% | 80.8% | 78.7% | 70.3% | 85.7% | 81.5% |
| Labels | 15.9% | 11.6% | 22.0% | 61.1% | 80.8% | 71.5% | 61.5% | 78.9% | 78.6% |

due to limited or no access to global resources by devices organized into peer-to-peer networks.

To accommodate these requirements and help move our research from controlled simulation environments to actual operational settings, we developed appropriate programming tools that enable online learning on edge devices. In particular, we developed a library, called Java Graph Neural Networks (JGNN – source code link in Subsection 4.2.7), which implements popular machine learning principles in Java. As such, it can be run on the billions of systems hosting the language's virtual machine (JVM)[8], such as micro-controllers and Android devices. Critically, JGNN is implemented on native Java and without external dependencies and hence mitigates application size. It can also run on devices that can execute wrapped lower-level binaries (e.g. precompiled processor or GPU instructions). Although the library does not account for GPUs, it supports parallelization.

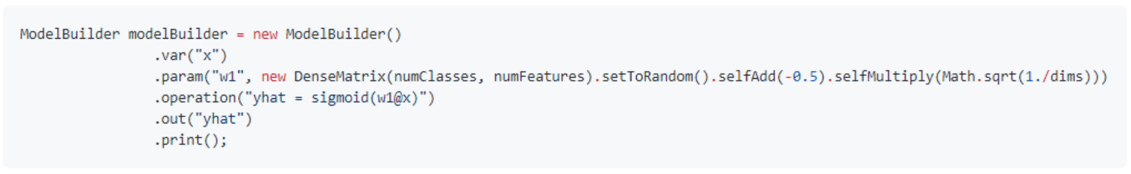
We expect this library to be used to train either small-scale models or local fragments of larger models distributed across devices. Hence, our main focus lies on minimizing memory footprints by providing tensor implementations that can directly expose subtensors. For example, it is possible to extract batches from training data and learn from those without allocating additional memory (save from a small overhead of creating wrapper objects and space pre-allocated for gradient storage). At the same time, we enable processor parallelization using Java's threading capabilities.

Furthermore, JGNN provides a true high-level abstraction of sparse representations that work interchangeably with dense matrices, a feature that traditional machine learning frameworks often struggle to provide, given the uncertainty of whether results should be sparse or dense. To work around this issue, we provide a conditional instantiation mechanism that automatically determines whether dense or sparse representations should be adopted depending on the order, data types of operands and which operation is performed, with plans to enrich this behavior with more policies, such as expected result sparsity. This way, AI systems can focus on defining their models and delegate data structure handling to the library instead of programmers developing them. It must be stressed that supporting sparse tensors is of paramount importance for the definition of GNNs, where using sparse matrices to represent graphs allow the matrix multiplications corresponding to graph convolutions to be computed in times that scale only linearly with the number of edges.

Finally, JGNN allows textual definitions of machine learning models, such as neural networks and GNNs, that are parsed by a model builder to appropriately instantiate machine learning primitives. This helps avoid lengthy definitions, for which Java code is notorious, but which do not suit the experimental needs of the machine learning community. For example, in Figure 2 we present how a simple logistic regression model can be defined in JGNN with only declaration of

[8]https://www.oracle.com/java/moved-by-java/timeline

its variables and formula and initialization of its parameter. Ongoing work in the library focuses on providing models and datasets out-of-the-box to facilitate ease of experimentation.



```
ModelBuilder modelBuilder = new ModelBuilder()
                .var("x")
                .param("w1", new DenseMatrix(numClasses, numFeatures).setToRandom().selfAdd(-0.5).selfMultiply(Math.sqrt(1./dims)))
                .operation("yhat = sigmoid(w1@x)")
                .out("yhat")
                .print();
```

*Figure 2. Definition of a logistic regression model using JGNN with input variable $x$, output variable yhat and initialized parameters $w1$.*

#### 4.2.6. Relevant preprints (to be submitted for publication)

- *E. Krasanakis, S. Papadopoulos, I. Kompatsiaris, A. Symeonidis.* pygrank: A Python Package for Graph Node Ranking, *arXiv preprint, arXiv:2110.09274, 2021* [103]
- *E. Krasanakis, S. Papadopoulos, I. Kompatsiaris.* p2pGNN: A Decentralized Graph Neural Network for Node Classification in Peer-to-Peer Networks, *arXiv preprint, arXiv:2111.14837, 2021* [92]

#### 4.2.7. Relevant software and/or external resources

- The JGNN library for native Java implementation of graph neural networks can be found in the public repository: `https://github.com/MKLab-ITI/JGNN`
- Implementation of decentralized graph neural networks and simulated experiments in Python supporting p2pGNN can be found in the public repository: `https://github.com/MKLab-ITI/decentralized-gnn`
- The gnn-test framework used to compare graph neural network approaches in Python over a tensorflow backend can be found in the public repository: `https://github.com/MKLab-ITI/gnn-test`

## 4.3. Towards efficient distributed training

**Contributing partners:** JR

### 4.3.1. Introduction

As can be seen from the evolution of popular distributed training frameworks like *DeepSpeed*[9], over time these "pick up" successful general strategies that improve training and port them to the distributed setting to also improve performance there. For example, during the evolution of DeepSpeed, popular adaptive gradient optimizer like Adam [104], and recently strategies from curriculum learning have been added[10], although only for the specific task of training Natural Language Processing (NLP) models. Similarly, JR follows a two-stage approach, where we first research novel general strategies for improving training and which can be ported efficiently to distributed training scenario, and will later on will port the strategies that have been identified

[9] `https://github.com/microsoft/DeepSpeed`
[10] `https://www.deepspeed.ai/tutorials/curriculum-learning/`

as successful to distributed systems. In the following, we describe our initial work, which deals with a novel “mini-batch trimming” strategy and belongs to the field of curriculum learning. This strategy improves the training (as can be seen in the experiments), and therefore we plan to port it to a distributed training framework in the future.

Training a neural network model which generalizes well (has good performance on unseen data) is a highly desirable property, but not easy to achieve. Nowadays, model training often employs adaptive gradient methods, such as the Adam optimizer [104], as they exhibit practical advantages (less sensitive to weight initialization and hyperparameters) compared with mini-batch Stochastic Gradient Descent (SGD). On the other hand, their generalization capabilities has been observed to not always be as good as SGD [105].

We propose a simple strategy which we call *mini-batch trimming* to increase the generalization capability (measured for image classification problems as the error of the final model on the test dataset) of trained models. The strategy is easy to integrate into existing training pipelines, does not need modification of model structures and is independent of the employed optimizer (can be used for both SGD and Adam-like methods). Its motivation lies in the fact that humans learn subjects (e.g. algebra) ‘from easy to hard’, i.e., we first learn basic concepts of the subject matter and only later move to more advanced ones. In the same way, we propose that optimizer should focus on the more difficult samples in the dataset only in the later training stages. For example, in the context of image classification, images that are harder to classify correctly will only be involved in the later stages of training.

Our strategy is similar to curriculum learning methods and importance sampling methods. In curriculum learning (see the survey in [106]), during training the samples are presented in a more meaningful order (e.g. from easy to hard) instead of the default random order. Importance sampling methods do not treat all samples in a dataset in the same way, but instead bias the selection of samples via a certain criterion. Characteristically, in [107] sampling is used to overweight highly representative samples during training. A disadvantage of this approach is that it has a complicated workflow, which involves density clustering (via the t-SNE algorithm [108]) in the sample space.

In the following subsections, we will describe our proposed mini-batch trimming strategy (Subsection 4.3.2) and present experiments on standard image classification problems which demonstrate that the strategy leads to models which generalize better (Subsection 4.3.3). We finally investigate another strategy (*random sample dropping* – Subsection 4.3.4) that did *not* provide benefits with respect to model generalization.

### 4.3.2. Mini-batch trimming

The training of a neural network model is usually done iteratively. In each iteration, a mini-batch consisting of $B$ samples (where $B$ is typically 64 or 128) is drawn randomly from the training set, the mean loss for the mini-batch is calculated in the forward pass and in the backward pass the gradient of the mean loss is utilized to update the model weights.

In order to focus more on the harder samples in the mini-batch, we propose a strategy which we call mini-batch trimming. As we cannot quantify the ‘hardness’ of a sample $\phi$ exactly, we take the per-sample loss $L(\phi)$ as an estimate of its hardness. This makes sense, as the more difficult samples in the training set typically also have a higher loss. We modify the forward pass now in the following way: First the per-sample loss $L(\phi)$ is calculated for all samples in the mini-batch. Next all samples in the mini-batch are sorted using the per-sample loss as criterion. The mean loss is now calculated only from a fixed fraction of the samples in the mini-batch with the highest per-sample loss. So we are calculating a kind of *trimmed mean* instead of the typical mean. For selecting the fraction $p$ of the samples with the highest loss the PyTorch framework provides the *torch.topk* operator, which is also differentiable. Example code demonstrating its usage can be seen in Listing 1.

```
# 'p' (in range [0, 1]) is the fraction of the samples
# in the mini-batch to be kept
# 'loss' is the loss vector containing the loss value
# for each sample in the mini-batch
# 'loss_trimmed' is the trimmed loss to be returned
loss_trimmed = torch.topk(loss, round(p * loss.size()[0]),
                          sorted = False, dim = 0)[0])
```

*Listing 1. Example python code showing how to implement mini-batch trimming.*

In this way, in each training iteration the update of the model weights is biased towards the more difficult samples. In the fashion of curriculum learning, the fraction $p$ is linearly decreased during training. For the first epoch $p$ has the value 1.0 (take all samples in mini-batch into account), whereas in the last epoch p is set to 0.2 (focus only on the 20 % samples in the mini-batch with the highest loss). Experiments have shown that this is a sensible choice.

Note that for neural network models without batch-normalization layers (e.g. transformer architectures for natural language processing), mini-batch trimming brings also a runtime improvement, as the backward pass then depends only on a part of the mini-batch. For models with batch-normalization layer (like most CNNs) this is not the case, because for these models the forward pass depends on the *whole* batch.

#### 4.3.3. Experiments and evaluation

For the experiments and evaluation, we employ three standard datasets for image classification: SVHN [11], CIFAR-10 and CIFAR-100 [12]. The datasets consist of 32x32 pixel RGB images, which belong to either 10 classes (SVHN and CIFAR-10) or 100 classes (CIFAR-100). We use the Adam optimizer, with learning rate set to 0.001 and weight decay set to 0.0001. The mini-batch size is 128 and training is done for 150 epochs, with the learning rate decayed by a factor of 0.5 at epochs 50 and 100. We perform the experiments with two popular neural network architectures for computer vision, Resnet-34 [109] and Densenet-121 [110].

To measure how well the trained model is able to generalize, we utilize the top-1 classification error of the final model on the test set (which of course has not been seen during training). For each configuration, we do 10 different runs with random seeds and take the average of these 10 runs. We compare the standard training with the variant with mini-batch trimming enabled. Results of the experiments can be seen in Table 3. The evaluation shows that mini-batch trimming is able to improve the generalization capability of the model in nearly all cases, except for one case (Densenet-121 architecture on CIFAR-10 dataset) where there is a slight decrease in the model performance.

#### 4.3.4. Random sample dropping

We researched also another novel strategy termed *random sample dropping*. This strategy works by *dropping* in each epoch randomly a certain percentage of samples, and filling the missing ones up by *sampling* randomly from the dataset (which is equivalent to sampling with replacement). This means that it is not guaranteed that each training sample is taken into account during the training step for one epoch. Instead, it may occur for that epoch that a specific training sample

[11] http://ufldl.stanford.edu/housenumbers/
[12] https://www.cs.toronto.edu/~kriz/cifar.html

Table 3. *Comparison of training with mini-batch trimming disabled/enabled for various network architectures anddatasets. The first value in each cell is the average test error (in percent, averaged over 10 runs) with mini-batchtrimming disabled, the second value is with mini-batch trimming enabled. The lower value is marked in bold.*

| **Dataset** | **ResNet-34** | **DenseNet-121** |
|---|---|---|
| SVHN | 5.87 / **5.76** | 4.62 / **4.42** |
| CIFAR-10 | 17.43 / **17.01** | **10.10** / 10.19 |
| CIFAR-100 | 48.19 / **47.72** | 32.95 / **32.18** |

is not used at all, or used exactly once, or even twice or more often. So we are increasing the variance in the set of training samples used in one epoch, which we assumed to be beneficial for the generalization of the trained network model. Unfortunately, preliminary evaluations on standard image classification datasets show that this strategy does not bring tangible benefits in terms of generalization and even slightly reduces the accuracy of trained models.

### 4.3.5. Relevant publications

- *H. Fassold*, Some like it tough: Improving model generalization via progressively increasing the training difficulty, *ASPAI, 2021*
  Zenodo record: https://zenodo.org/record/5596848 [111]

### 4.3.6. Relevant software and/or external resources

- A Python implementation of mini-batch trimming can be found on the public repository: https://github.com/hfassold/mbtrim

## 4.4. Resource aware models for the edge

**Contributing partners:** IDF

### 4.4.1. Resource adaptable RNN

Recurrent Neural Networks (RNNs) are neural networks that process sequences of inputs, such as frames of videos. Existing approaches for RNNs on limited computational resources typically focus on minimizing resource usage and their overwhelming majority try to maintain accuracy or reach a set efficiency. We, on the other hand, are interested in constructing models that can adapt to changing resource constraints. ThRNN [77] is a RNN architecture that aims to control computation time by not processing some inputs. The trade-off between the average accuracy and the average number of updates can be modified during inference by changing a single parameter $thr$. However, ThrRNN optimizes computational cost on average over sequences. It is therefore unable to meet strict constraints that would prevent models from exceeding computational budgets. Consequently, optimized models can shut down or exhibit output delays. On the other hand, we present a new approach, called Skip-Windows (SkipW), that tackles this problem by strictly enforcing computational constraints over subsequences of inputs.

In a nutshell, SkipW forces RNNs to process at most $K$ out of every non-overlapping sequence of $L$ inputs. It can be wrapped around any RNN cell $S$. It uses a conditional computation mechanism to skip some updates. Before any new L-size window of inputs, a L-size vector $\tilde{u}_W$ is computed. $\tilde{u}_W[i]$ can be seen as the importance of input $i$ in the window. Then, the architecture includes a

$select_K$ mechanism. This function takes as input the vector $\tilde{u}_W$ and outputs the vector $\tilde{u}_W^K$, setting $L-K$ elements to a value that ensures the associated inputs are not processed. Therefore, it forces the RNN cell to skip $(L-K)$ out of every $L$ inputs. Appropriately selecting $K$ ensures a strict upper bound on the computational cost of the model for a sequence and for each subsequence. As an example, the inputs associated to the $K$ highest values in $\tilde{u}_W$ can be processed. This architecture is illustrated in Figure 3.

In terms of accuracy, SkipW compares well to existing methods. To show this, we evaluated it on a Human Activity Recognition (HAR) from video task. The accuracy and computational cost (measured in terms of input processed) is reported for various approaches and configurations in Figure 4. It can be seen that large window sizes offer good trade offs when changing $K$. For example, for $L = 16$, reducing $K$ from 4 to 1 more than halves the number of inputs processed for a drop of 3.6% in accuracy. The trade-offs achieved by SkipW are better than what could be achieved with ThrRNN (discussed above) and SkipRNN, another baseline that subsamples inputs but cannot be adapted in real time. This figure also includes a comparison with two naive subsampling strategies selecting a set number of inputs. *Random subsampling* samples inputs based on a uniform distribution. *Periodic subsampling* selects them at regular interval. Both naive strategies are worse than SkipW.

SkipW was also better or similar to the baselines on two other data sets (Adding Task and IMDB) while it was worse than non-adaptive models but better than ThrRNN on a fourth one (sequential MNIST). We have also implemented SkipW on small hardware to showcase its performance in the setting it is designed for. We implement the full HAR service, from images to activity recognition, on two Nvidia Jetson platforms. Experiments confirm that SkipW can lead to faster processing and to energy savings. The latter is shown in Figure 5.

### 4.4.2. Inplace knowledge distillation with teacher assistant

A popular strategy to construct resource-adaptable deep neural networks is to construct a set of nested models [72, 73, 74, 75, 112, 113, 114]. In other words, such a flexible model is constituted of several models (one per resource/performance operating point) that are embedded one into another like Matryoshka dolls with strong parameters sharing: one largest model and several sub-models. At the inference, a suitable sub-model corresponding to the available resources may be instantly extracted and deployed, and, thanks to the strong parameters sharing, the full model

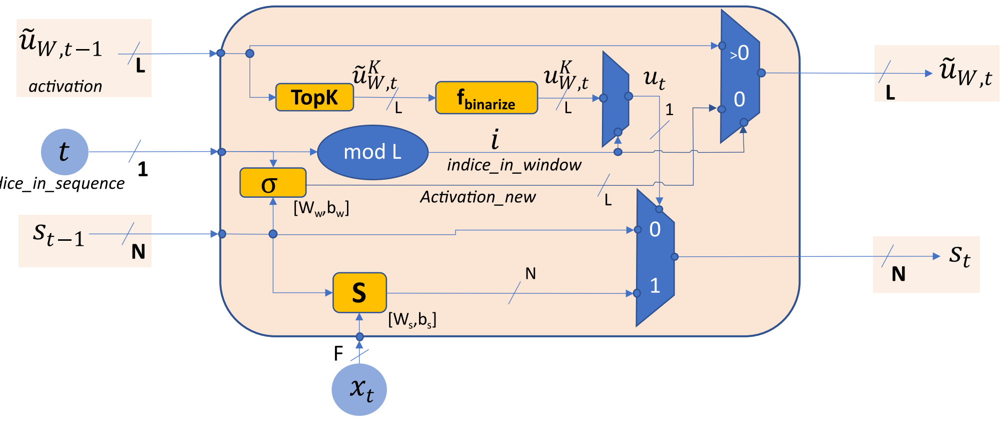


*Figure 3. Skip-Window cell implementation example.*

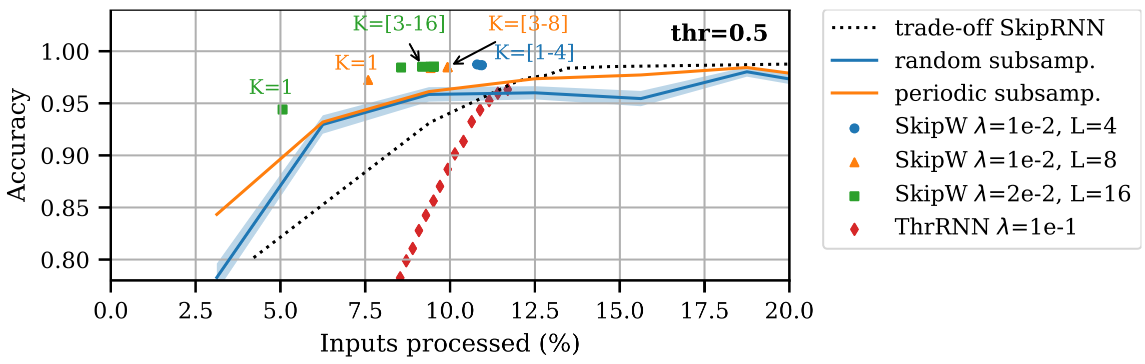


*Figure 4. SkipW achieves both a better accuracy and a lower computational cost than baselines on a HAR task. For random subsampling, the shaded area corresponds to 3 times the standard deviation on each side of the mean value (50 evaluations).*

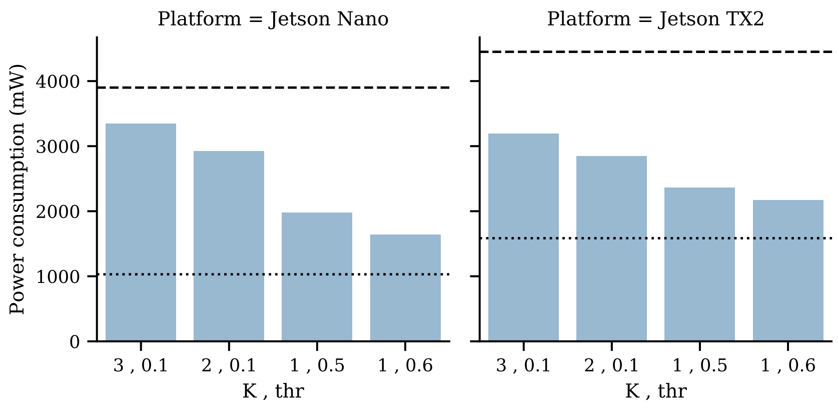


*Figure 5. Average energy consumption for the analysis of a HAR sequence using SkipW and PoseNet (MobileNet 0.75) on a Jetson Nano and a Jetson TX2. The dotted line corresponds to the energy level for no activity and the dashed one to the maximum instantaneous level measured when models are running.*

may be efficiently transmitted and stored, as compared to a dummy solution of training several independent models (one per resource/performance operating point). We have developed a better approach to train such nested models.

Let flexible DNN include $n$ sub-models enumerated in the order of their sizes (the largest network is indexed by $n$). Let vector $a_s[i]$ or $a_t[i]$ be the logits of the $i$-th model ($i = 1, \dots, n$). The most conventional way to train the flexible DNN consists in a supervised joint learning of all sub-networks as:

$$\mathcal{L}^{flex} = \sum_{i=1}^{n} \mathcal{L}_{CE}(a_s[i], y_r), \tag{9}$$

where $\mathcal{L}_{CE}(\cdot, \cdot)$ is the cross-entropy loss.

It was proposed in [74] to still train the sub-models jointly, but not all of them in a completely supervised manner: the largest $n$-th model is trained in a supervised way, while all other sub-models are distilled from the largest model. The resulting in-place knowledge distillation (IPKD) optimization loss writes

$$\mathcal{L}^{flex}_{IPKD} = \mathcal{L}_{CE}(a_s[n], y_r) + (1-\lambda)\sum_{i=1}^{n-1}\mathcal{L}_{CE}(a_s[i], y_r) + \lambda\sum_{i=1}^{n-1}\mathcal{L}_{KD}(a_s[i], a_t[n]), \tag{10}$$

where

$$\mathcal{L}_{KD}(a_s, a_t) = \tau^2\mathcal{D}_{KL}\left(\text{softmax}(a_s/\tau), \text{softmax}(a_t/\tau)\right) \tag{11}$$

and $\mathcal{D}_{KL}$ is the Kullback-Leibler (KL) divergence. *In-place* in IPKD refers to the fact that the KD is now performed jointly with a strong parameters sharing between the sub-models. It was shown in [74] for slimmable networks that the IPKD training outperforms the conventional training (9). The IPKD strategy was then adopted in [72, 75, 113, 114].

In this work, we build on the idea of KD with teacher assistant [115] to improve the IPKD training of flexible models. Indeed, in IPKD all sub-models are distilled from the biggest one. It was remarked in [115] that when the gap in model size between the teacher and the student is big enough, distilling the knowledge directly from the teacher might be sub-optimal. To overcome this issue, the authors of [115] introduced an intermediate *teacher assistant* model that is first learned by the teacher and then teaches the student. This approach has been shown [115] to lead to a better student model performance. Based on this approach, we propose two methods to train flexible models.

We first introduce the IPKD with one Teacher Assistant (IPKD-TA-1), where each sub-model is taught by another sub-model that is just next to it from the top. This leads to the following re-formulation of IPKD loss (10):

$$\mathcal{L}^{flex}_{IPKD-TA-1} = \mathcal{L}_{CE}(a_s[n], y_r) + (1-\lambda)\sum_{i=1}^{n-1}\mathcal{L}_{CE}(a_s[i], y_r) + \lambda\sum_{i=1}^{n-1}\mathcal{L}_{KD}(a_s[i], a_t[i+1]). \tag{12}$$

Another strategy we introduce and investigate is the IPKD with multiple teacher assistants (IPKD-TA-M), where each sub-model is taught by all the larger sub-models [13]. This is achieved by writing the corresponding loss as follows:

$$\begin{aligned}\mathcal{L}^{flex}_{IPKD-TA-M} = \mathcal{L}_{CE}(a_s[n], y_r) + (1-\lambda)\sum_{i=1}^{n-1}\mathcal{L}_{CE}(a_s[i], y_r) + \\ + \lambda\sum_{i=1}^{n-1}\frac{1}{n-i}\sum_{j=i+1}^{n}\mathcal{L}_{KD}(a_s[i], a_t[j]),\end{aligned} \tag{13}$$

where the weights $\frac{1}{n-i}$ (such that $\sum_{j=i+1}^{n}\frac{1}{n-i} = 1$) are introduced in order to re-balance the impacts of the teacher assistants, since the number of teacher assistants varies from one sub-model to another.

We evaluated these approaches on several flexible models and data sets. Figure 6 is a representative example, obtained using MSDNet [112] on the CIFAR-10 data set. This particular model can produce an output at multiple locations (called exit points) in the network. The resulting accuracy is reported for every such exit point in the network. We may see that, as compared to the baseline supervised training without distillation, the investigated IPKD training improves the results.

[13] In general each sub-model can be taught by a subset of larger sub-models.

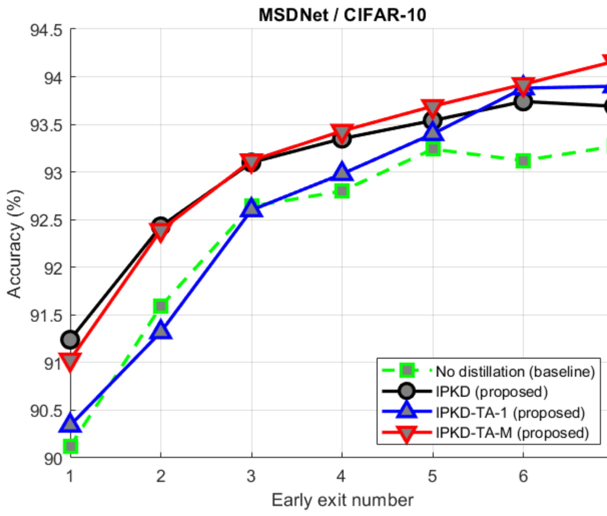


*Figure 6. Results in terms of classification accuracy for the proposed training approach, using a MSDNet model on CIFAR-10.*

### 4.4.3. Relevant publications

- *T. Mayet, A. Lambert, P. Le Guyadec, F. Le Bolzer, and F. Schnitzler.* SkipW: Resource adaptable RNN with strict upper computational limit, *ICLR, 2020* [116]
  Zenodo record: `https://zenodo.org/record/4911371`
- *A. Ozerov and N. Q. K. Duong.* Inplace knowledge distillation with teacher assistant for improved training of flexible deep neural networks, *EUSIPCO, 2021* [117]
  Zenodo record: `https://zenodo.org/record/5047247`

# 5. Quantum Assisted Reinforcement Learning (Task 3.8)

In recent years, the technological capability to control a single Quantum system has opened the possibility of using these systems as computers. With this approach, we have access to a novel model of computation, which for some tasks has been proven to outperform conventional algorithms. While the full implementation of a Quantum computer is still far from being a complete reality, preliminary devices are already operational in a number of physics laboratories. Some private companies allow remote access to this technology, allowing researches to experiment with novel approaches to established computational problems.

Among relevant applications of Quantum computers, Quantum Machine Learning studies the utility of these devices to process information and perform training and generative tasks. The computational requirements of Machine Learning motivate this approach, as the computational advantage of Quantum resources could boost fundamental operations.

The research performed in this task contributes to the field of Quantum Machine learning, and brings insight into applications of Quantum computers to perform Machine Learning with either classical or Quantum data. In particular, it focuses in fundamental operations designed as algorithms executed on Quantum computers. Through a collaboration with experimental partners, the developments of this tasks are demonstrated on real devices, going beyond the stage of theoretical proof of concept.

## 5.1. Overview of quantum assisted reinforcement learning contributions

A single-qubit circuit can approximate any bounded complex function stored in the degrees of freedom defining its quantum gates. The single-qubit approximant presented in this work is operated through a series of gates that take as their parameterization the independent variable of the target function and an additional set of adjustable parameters. The independent variable is re-uploaded in every gate, while the parameters are optimized for each target function. The output state of this quantum circuit becomes more accurate as the number of re-uploadings of the independent variable increases, i.e., as more layers of gates parameterized with the independent variable are applied. In this work, we provide two different proofs of this claim related to both Fourier series and the Universal Approximation Theorem (UAT) for neural networks, and we benchmark both methods against their classical counterparts. We further implement a single-qubit approximant in a real superconducting qubit device, demonstrating how the ability to describe a set of functions improves with the depth of the quantum circuit. This work shows the robustness of the re-uploading technique on Quantum Machine Learning.

## 5.2. Universal approximation of functions with Quantum computers

**Contributing partners:** BSC

### 5.2.1. Introduction

We present two independent proofs that any bounded complex function can be approximated in a convergent way by a quantum circuit acting on one qubit, constituting a single-qubit universal approximant. The two methods derived allow using a Quantum circuit to approximate arbitrary mathematical functions, and provide two constructive methods to build the corresponding Quantum circuit. This demonstrates the precise representation power of a single-qubit circuit, which increases as more layers are added. The essential element of the present construction is the re-uploading of the input variables along the action of the quantum gates. Thus, in analogy to neural networks, query complexity is attached to accuracy. The first way to prove this result is to make contact with harmonic analysis. This is a natural step as single-qubit gates are expandable in Fourier series that can be rearranged to fit existing theorems. The second method is analogous to the UAT using a translation into quantum circuits. A series of specific gates leads to an output state that approximates functions uniformly. In both cases, the quantum theorems inherit the applicability and characteristics of their classical counterparts.

We provide numerical benchmarks of these theorems computed via classical simulation of quantum computers. Our simulations show how an increasing query complexity can improve the accuracy of the approximation of a number of test functions. The way a single-qubit circuit can approximate any function in an experimental setup by using a superconducting qubit is explicitly illustrated. Experimental results confirms the trend of simulations up to a point where the accumulation of errors dominates the experiment.

### 5.2.2. One qubit as an approximant of mathematical functions

We present two sets of single-qubit gates to construct quantum circuits that represent arbitrary complex functions. Each set is based on known results from the theory of function approximations, namely Fourier series and Universal Approximation Theorem (UAT), respectively. The range of applicability of these theorems for quantum circuits and the conditions for universality are thus inherited from their classical counterparts.

We present two Quantum version of the Fourier series approximation theorem, and the Quantum equivalent to the Universal Approximation Theorem (UAT):

**Quantum Fourier series.** Let $f, \phi$ be any pair of functions $f : \mathbb{R} \to [0, 1]$ and $\phi : \mathbb{R} \to [0, 2\pi)$ , such that $z(x) = f(x)e^{i\phi(x)}$ is a complex function with a finite number of finite discontinuities integrable within an interval $[a, b] \in \mathbb{R}$ of length $P$. Then, there exists a set of parameters $\{\vec{\theta}_1, \vec{\theta}_2, \ldots, \vec{\theta}_N\}$ such that

$$\langle 1| \prod_{i=1}^{N} U^{\mathcal{F}}(x, \vec{\theta}_i)|0\rangle = z_N(x), \tag{14}$$

with $z_N(x)$ the $N$-terms Fourier series.

**Quantum UAT.** Let $f, \phi$ be any pair of functions $f : I_m \to [0, 1]$ and $\phi : I_m \to [0, 2\pi)$ , such that $z(\vec{x}) = f(\vec{x})e^{i\phi(\vec{x})}$ is a complex continuous function on $I_m$, with $I_m = [0, 1]^m$. Then there is an integer $N$ and a set of parameters $\{\vec{\theta}_1, \vec{\theta}_2, \ldots, \vec{\theta}_N\}$ such that

$$\left| f(\vec{x})e^{i\phi(\vec{x})} - \langle 1| \prod_{i=1}^{N} U^{\mathrm{UAT}}(\vec{x}, \vec{\theta}_i)|0\rangle \right| < \epsilon, \tag{15}$$

for any $\epsilon > 0$.

### 5.2.3. Experiments

We implement the single-qubit universal approximant in a superconducting qubit circuit cooled to the base temperature of a dilution refrigerator (20mK). The qubit is a 3D transmon geometry located inside an aluminum three-dimensional cavity. The cavity bare frequency, $\omega_c = 2\pi \times 7.89$ GHz, is greatly detuned from the qubit frequency, $\omega_q = 2\pi \times 4.81$ GHz. Hence, there is a qubit state-dependent dispersive shift on the cavity resonance, $2|\chi| = 2\pi \times 1.5$MHz. The qubit anharmonicity is $\alpha = -2\pi \times 324$ MHz and the qubit relaxation and spin-echo decay times are, respectively,

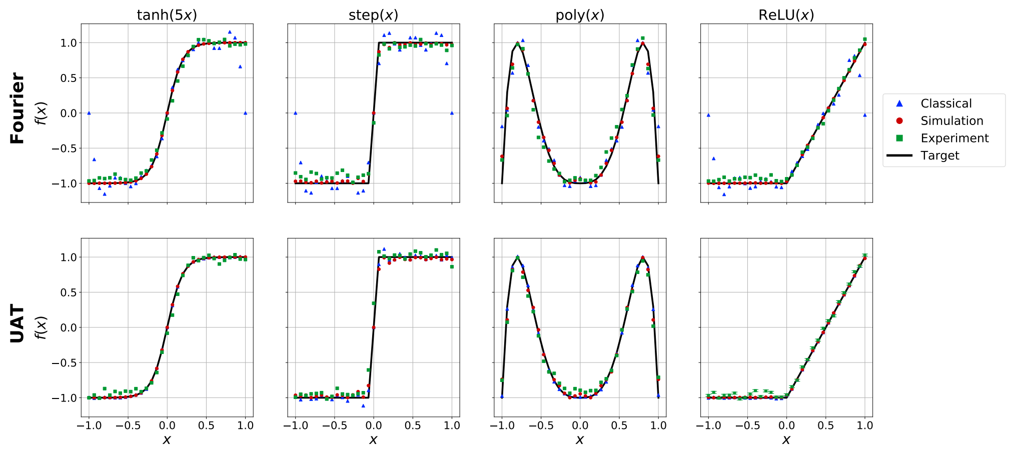


*Figure 7. Fittings for four real-valued functions using the Z benchmark with five layers. Blue triangles represent classical models, namely Fourier and UAT, while red dots represent its quantum counterparts computed using a classical simulator. Green squares are the experimental execution of the optimized quantum model using a superconducting qubit. The target function is plotted in black for comparison. The analysis for experimental errors is plotted for the* ReLU *function and the UAT model.*

$T_1 = 15.6\ \mu s$ and $T_{2Echo} = 12.0\ \mu s$. These time scales exceed the operation times needed to implement the algorithm up to 6 layers by 2 orders of magnitude.

An optimization process is required to find the optimal gate parameters. For quantum methods, optimization brings more problems that are yet to be solved. In particular, the landscape of the loss function in the parameter space remains unknown, and thus it is hard to infer what kind of classical optimizers perform well for each particular problem. For this reason we look for the best-fit parameters using the aforementioned `L-BFGS` algorithm and the genetic option `CMA`. Genetic algorithms explore vast regions of the parameter space and do not depend on gradients. However, they usually require more function evaluations to converge to the minimum. In this case it is not possible to guarantee that the solution found by any optimization algorithm is the global minimum of our loss function.

In Figure 7 we show the resulting fit for four single-variable real-valued functions of interest (see caption). In this case the $Z$ benchmark with 5 layers is considered. A classical approximation (blue), a quantum exact simulation (red) and its experimental implementation (green) are depicted. All methods follow the overall shape of the target function. Classical Fourier approximations return less accurate predictions on the value of $f(x)$ due to the periodic nature of the model. The quantum Fourier and both classical and quantum UAT models return better results for all values of $x$. This behaviour is observed in all benchmarks. The experimental results retain the qualitative properties of the exact models, although a loss in performance is visible. In addition, an analysis of experimental uncertainties is also depicted at the UAT ReLU plot from Figure 7.

We have shown that a single-qubit circuit has enough flexibility to encode any complex function $z(x)$ in the degrees of freedom of each quantum gate. This universal representation is achieved by acting with a quantum circuit on a single-qubit gate that depends on input variables as well as additional parameters that are fixed by Machine Learning techniques.

This result guarantees that a single-qubit circuit, as defined in this work, is able to store two different and independent real functions. These functions are not restricted to be single-variable, as there exists no limitation to the dimensionality of its independent variable. Our present results provide the highest degree of compression of data in a single-qubit state, since there are no more degrees of freedom available in a qubit.

The proof for universality was shown following two different approaches, leading to two sets of single-qubit gates. In the first method, we found a link between quantum circuits and Fourier series. We have defined a quantum gate tuned by five parameters such that the row of $N$ gates applied to an initial state provides a final state where an $N$-term Fourier series is encoded. For the second method, a single-qubit quantum gate is applied multiple times to yield a final state whose form is compatible with the Universal Approximation Theorem. The input state does not compromise the validity of the approximation theorems but it affects the parameters defining the circuit.

We also provide numerical evidence on the flexibility and approximation capabilities of these quantum circuits. The benchmarks have been obtained using simulations and classical minimizers to find optimal parameters for a set of test functions. We have included as benchmarks 1D and 2D real functions and 1D complex functions. The final results have also been compared to its classical counterparts. In all cases, it is possible to see an equivalent scaling for both classical and quantum methods. This ensures numerically that the quantum procedure is comparable to the standard classical ones.

Experimental results implemented using a superconducting trasmon qubit confirm the same trend obtained with the classical simulations. The finite qubit coherence does not seem to impact the results significantly in the gate sets applied.

### 5.2.4. Relevant publications

- *Adrián Pérez-Salinas, David López-Núñez, Artur García-Sáez, P. Forn-Díaz, and José I. Latorre.* One qubit as a universal approximant, *Phys. Rev. A 104, 012405 (2021)* [118]
  Zenodo record: https://zenodo.org/record/5531271

### 5.2.5. Relevant software and/or external resources

- Code related to the paper *One qubit as a Universal Approximant* can be found in the repository: https://github.com/UB-Quantic/Universal-Approximator

# 6. Ongoing Work and Conclusions

## 6.1. Relation to WP8 Use Cases

Research outcomes presented in this deliverable support AI by speeding up hyperparameter selection procedures (Section 3), deploying systems to non-centralized settings (Section 4) or taking advantage of dedicated quantum hardware for fast computations (Section 5). Contributions can be used to create accelerated or non-centralized versions of a wide range of AI systems, including those described in the use cases of D8.1. In this subsection, we outline promising ways in which research results can improve use case features.

First, faster neural model selection, distributed learning and quantum computing can be used in the context of use case feature 4B5 *"Efficiency"* to reduce infrastructure overhead when tuning AI and, if distributed learning clusters or quantum computers are available, speed up training and inference time by replacing traditional computing units. Speed improvements can also be used by use case feature 3C2 *"Just-in-time content creation & adaptation"* to train and execute novel event detection systems with new data and use them to notify journalists.

Of the non-centralized computing paradigms, decentralized graph neural networks could be used for deployment of mobile device tools, such as content verification systems satisfying use case features 1A1 *"Synthetic Text Detection/Verification"* and 1A2 *"Synthetic Image Detection/Verification"* that are trained based on real-world activity without the upkeep cost of centralized services. In a future scenario, we envision usage of decentralized learning in mobile social networks that gather user annotations of deepfakes and use these to collectively annotate incoming content, such as text and images, as fake or not while preserving the confidentiality of input content (which is often crucial, due to privacy or security constraints). Furthermore, resource-aware models executing at the edge adhere to localized data processing of use case feature 7D2 *"On-premise processing"* by respectively enabling collective learning of locally processed data and deploying AI for predictions.

Finally, partner contributions include new software tools that implement research breakthroughs. The source code has been made publicly available through code repositories, so that owners of research infrastructure using these tools can easily assert that implementations are straightforward per use case feature 4B3 *"Source code accessibility"*. Moreover, the source code can be downloaded and applied on AI solutions that are trained and deployed within institutional infrastructure per the use case feature 4B1 *"Local execution"*. Source code is in large part written in Python per use case feature 4B4 *"Runtime environment"*.

## 6.2. Ongoing and future work

Below, we summarize ongoing and future work of each task, organized per contributors.

### 6.2.1. Neural architecture search (Task 3.4)

**UNITN** will focus on investigating neuro-evolutionary network architecture search approaches, which will automatically search for optimal network architectures using evolutionary algorithms. Although these approaches are very effective, evolutionary algorithms rely heavily on having a large population of individuals (i.e., network architectures) and are therefore memory expensive. As such, we will look at regularized evolutionary algorithms with low memory footprint to evolve a dynamic image classifier. In details, we will consider novel custom operators that regularize the evolutionary process of a micro-population of a few individuals.

### 6.2.2. AI at the edge, decentralised and distributed learning (Task 3.5)

**CERTH** will port more AI models to decentralized and at the edge settings and will continue evolving developed frameworks, for example to facilitate learning with non-local graph diffusion schemes when homophily is not a valid hypothesis and support multiple samples per device. Software outcomes will also be refined to reach a state of technological readiness that will support easy integration of AI solutions in real-world decentralized applications by non-experts. These developments will be guided by the aforementioned prospective deployment in use case features 1A1 *"Synthetic Text Detection/Verification"* and 1A2 *"Synthetic Image Detection/Verification"*.

Moreover, distributed algorithms will be developed that account for non-IID data, as well as privacy constraints, and their efficacy will be demonstrated within the context of synthetic audio detection of WP8's use case feature 1A3 *"Synthetic Audio Detection/Verification"*. In detail an audio dataset will be extracted from the open-source TIMIT Corpus [119] of human audio recordings, supplemented with fake recordings produced by state-of-the-art text-to-speech algorithms, and split across several simulated devices under non-IID distributions. Then, federated learning techniques will be used to train deep neural models of fake audio detection without sharing data and hence adhering to legal contexts and on-premise processing requirements of use case features 1E1 *"Information about Legal Data Compliance"* and 7D2 *"On-premise processing"* respectively.

**JR** will research more parameter update algorithms that can be used in the training of distributed AI models. In detail, there is ongoing research on the novel *AdaFamily* algorithm, a family of adaptive gradient optimization algorithms resembling Adam, AdaBelief and AdaMomentum. These will be compared to mini-batch trimming to identify the optimizers with the best generalization capabilities in heterogeneous distributed computing. Finally, both the AdaFamily and mini-batch trimming will be implemented within distributed training frameworks, such as DeepSpeed and Pytorch Lightning.

### 6.2.3. Quantum assisted reinforcement learning (Task 3.8)

**BSC** Development on Quantum algorithms for the approximation of mathematical functions allows the implementation of arbitrary operations into a Quantum circuit. From these circuits, further development will follow in order to implement more complex operations as a combination of the approximation of elementary mathematical functions. These algorithms are then intended to be used in complex optimization problems and learning operations on classical data.

In addition, planned research aims to extend the application of arbitrary mathematical functions to complex operations performed by a Quantum device. In particular, operations related to the interaction between a Quantum environment (such as a Quantum device in the lab) and a controlling device using past development of arbitrary functions.

## 6.3. Conclusions

In this deliverable, we presented the initial research and development results of WP3 regarding decentralized and high-performance learning that were obtained by Task 3.4 (Neural Architecture Search), Task 3.5 (AI at the Edge, Decentralised and Distributed Learning) and Task 3.8 (Quantum Assisted Reinforcement Learning). These tasks explored novel directions of scaling computing power, i.e., respectively through algorithmically efficient neural architecture search, by sharing the burden of computations between multiple (edge) workstations and devices, and with quantum computing that reduces running time complexity of popular machine learning operations. The efficacy of developed approaches was tested on a variety of popular AI tasks, such as image classification, social network mining and text classification. Beyond the scope of testing, explored principles are generic and can support AI on other kinds of media data too.

In terms of architecture search in T3.4, a workshop and a journal special issue were organized, a survey on recent developments was performed and a new algorithm was developed to automate search with few-shot learning. New methods to share the burden of AI model calculations across multiple devices were explored in T3.5; these span decentralized learning of model fragments to classify in-device data, distributed training of large datasets and ways to compress models to fit on available resources, where all directions improve existing non-centralized computing paradigms when run on distributed workstations or edge devices. Finally, T3.8 explored quantum computing acceleration of machine learning operations by showing how its backbone universal approximation theorem can be achieved with the highest possible compression.

Reported task activities comprise domain exploration, progressing state-of-the-art research in terms of theoretical and experimental results and providing publicly available software tools for practical application. Overall, there is a satisfactory number of outcomes until this point. These comprise 1 workshop, 1 special issue, 2 journal publications, 4 conference publications, 2 preprints to be submitted for peer review in the near future, and 5 software projects that implement existing research and can support practical deployment and ongoing research. The outcomes of ongoing and future work detailed in this deliverable will be reported in D3.4 (due in M48).

A future goal for our work is to foster joint research that can prove mutually beneficial to cooperating partners both within WP3 and in conjunction with other work packages. For example it could be of interest to cross-breed non-centralized AI with other learning paradigms investigated in WP3 or research outcomes of other work packages. At the same time, it is of high research interest to apply architecture search or model compression practices to decentralized learning, where model fragments may use different models based on local data or resources.

## References

[1] Y. LeCun, J. Denker, and S. Solla, “Optimal brain damage,” in *Proc. NeurIPS*, 1989.

[2] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural networks,” in *Proc. NeurIPS*, 2015.

[3] S. Han, H. Mao, and W. Dally, “Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding,” in *Proc. ICLR*, 2016.

[4] Y. Guo, A. Yao, and Y. Chen, “Dynamic network surgery for efficient dnns,” in *Proc. NeurIPS*, 2016.

[5] G. Li, C. Qian, C. Jiang, X. Lu, and K. Tang, “Optimization based layer-wise magnitude-based pruning for dnn compression,” in *Proc. IJCAI*, 2018.

[6] X. Dai, H. Yin, and N. K. Jha, “Nest: a neural network synthesis tool based on a grow-and-prune paradigm,” *IEEE Trans. Comput.*, vol. 68, no. 10, p. 1487–1497, 2019.

[7] S. Anwar, K. Hwang, and W. Sung, “Structured pruning of deep convolutional neural networks,” *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, p. 1–18, 2017.

[8] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf, “Pruning filters for efficient convnets,” in *Proc. ICLR*, 2017.

[9] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, “Network trimming: a data-driven neuron pruning approach towards efficient deep architectures,” *arXiv preprint arXiv: 1607.03250*, 2016.

[10] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, “Soft filter pruning for accelerating deep convolutional neural networks,” in *Proc. IJCAI*, 2018.

[11] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in *Proc. NeurIPS*, 2016.

[12] J.-H. Luo, J. Wu, and W. Lin, “Thinet: a filter level pruning method for deep neural network compression,” in *Proc. ICCV*, 2017.

[13] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” in *Proc. ICCV*, 2017.

[14] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, “Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers,” in *Proc. ICLR*, 2018.

[15] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. Davis, “Nisp: pruning networks using neuron importance score propagation,” in *Proc. CVPR*, 2018.

[16] J. Frankle and M. Carbin, “The lottery ticket hypothesis: finding sparse, trainable neural networks,” in *Proc. ICLR*, 2019.

[17] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” in *Proc. ICLR*, 2019.

[18] Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing deep convolutional networks using vector quantization,” *arXiv preprint arXiv:1412.6115*, 2014.

[19] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. CVPR*, 2016.

[20] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. ICML*, 2016.

[21] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: training deep neural networks with binary weights during propagations," in *Proc. NeurIPS*, 2015.

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: imagenet classification using binary convolutional neural networks," in *Proc. ECCV*, 2016.

[23] Q. Hu, P. Wang, and J. Cheng, "From hashing to cnns: training binary weight networks via hashing," in *Proc. AAAI*, 2018.

[24] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[25] C. Zhu, S. Han, H. Mao, and W. Dally, "Trained ternary quantization," in *Proc. ICLR*, 2018.

[26] M. Kim and P. Smaragdis, "Bitwise neural networks," *arXiv preprint arXiv:1601.06071*, 2016.

[27] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. NeurIPS*, 2016.

[28] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. BMVC*, 2014.

[29] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, p. 1943–1955, 2015.

[30] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *Proc. ICLR*, 2014.

[31] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," in *Proc. ICLR*, 2015.

[32] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. NeurIPS*, 2014.

[33] A. Romero, N. Ballas, S. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: hints for thin deep nets," in *Proc. ICLR*, 2015.

[34] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer," in *Proc. ICLR*, 2017.

[35] A. Malinin, B. Mlodozeniec, and M. Gales, "Ensemble distribution distillation," in *Proc. ICLR*, 2020.

[36] Y. Shen, Z. Zhang, M. Sabuncu, and L. Sun, "Real-time uncertainty estimation in computer vision via uncertainty-aware distribution distillation," in *Proc. WACV*, 2021.

[37] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Proc. ICLR*, 2016.

[38] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proc. ICCV*, 2017.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.

[40] G. Huang, Z. Liu, L. V. D. Maaten, and K. Weinberger, "Densely connected convolutional networks," in *Proc. CVPR*, 2017.

[41] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. ICLR*, 2014.

[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. CVPR*, 2016.

[43] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[44] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. CVPR*, 2017.

[45] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: an extremely efficient convolutional neural network for mobile devices," in *Proc. CVPR*, 2018.

[46] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: practical guidelines for efficient cnn architecture design," in *Proc. ECCV*, 2018.

[47] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[48] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: inverted residuals and linear bottlenecks," in *Proc. CVPR*, 2018.

[49] X. Bai, X. Liu, Q. Liu, J. Song, N. Sebe, and B. Kim, "Explainable deep learning for efficient and robust pattern recognition: A survey of recent developments," *Pattern Recognition*, vol. 120, Article 108102, 2021.

[50] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[51] M. Patacchiola, J. Turner, E. J. Crowley, M. O' Boyle, and A. J. Storkey, "Bayesian meta-learning for the few-shot setting via deep kernels," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 16108–16118, Curran Associates, Inc., 2020.

[52] L. S.-T. Martin Wistuba, Nicolas Schilling, "Two-stage transfer surrogate model for automatic hyperparameter optimization.," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy*, 2016.

[53] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, p. 281–305, Feb. 2012.

[54] M. Feurer, B. Letham, F. Hutter, and E. Bakshy, "Practical transfer learning for bayesian optimization," 2021.

[55] M. Volpp, L. P. Fröhlich, K. Fischer, A. Doerr, S. Falkner, F. Hutter, and C. Daniel, "Meta-learning acquisition functions for transfer learning in bayesian optimization," in *International Conference on Learning Representations*, 2020.

[56] V. Perrone, R. Jenatton, M. W. Seeger, and C. Archambeau, "Scalable hyperparameter transfer learning," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[57] M. Wistuba and J. Grabocka, "Few-shot bayesian optimization with deep kernel surrogates," *arXiv preprint arXiv:2101.07667*, 2021.

[58] V. Marx, "The big challenges of big data," *Nature*, vol. 498, no. 7453, pp. 255–260, 2013.

[59] H. R. Varian, "Beyond big data," *Business Economics*, vol. 49, no. 1, pp. 27–31, 2014.

[60] A. Taherkordi, F. Eliassen, and G. Horn, "From iot big data to iot big services," in *Proceedings of the Symposium on Applied Computing*, pp. 485–491, 2017.

[61] H. Liu, "Beyond the scale of big data," *Frontiers in big Data*, vol. 1, p. 1, 2018.

[62] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," *arXiv preprint arXiv:1301.0159*, 2013.

[63] R. M. Alguliyev, R. T. Gasimova, and R. N. Abbaslı, "The obstacles in big data process.," *International Journal of Modern Education & Computer Science*, vol. 9, no. 3, 2017.

[64] A. C. Djedouboum, A. A. Abba Ari, A. M. Gueroui, A. Mohamadou, and Z. Aliouat, "Big data collection in large-scale wireless sensor networks," *Sensors*, vol. 18, no. 12, p. 4474, 2018.

[65] M.-L. Song, R. Fisher, J.-L. Wang, and L.-B. Cui, "Environmental performance evaluation with big data: Theories and methods," *Annals of Operations Research*, vol. 270, no. 1, pp. 459–472, 2018.

[66] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*, pp. 1–19, Springer, 2008.

[67] A. Friedman and A. Schuster, "Data mining with differential privacy," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 493–502, 2010.

[68] C. Dwork, A. Roth, *et al.*, "The algorithmic foundations of differential privacy.," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[69] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *CoRR*, vol. abs/1711.07128, 2017.

[70] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

[71] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 331–344, IEEE, 2019.

[72] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020.

[73] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *International Conference on Learning Representations*, 2019.

[74] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1803–1811, 2019.

[75] L. Guerra, B. Zhuang, I. Reid, and T. Drummond, "Switchable precision neural networks," *arXiv preprint arXiv:2002.02815*, 2020.

[76] Q. Jin, L. Yang, and Z. Liao, "Adabits: Neural network quantization with adaptive bit-widths," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2146–2156, 2020.

[77] A. Lambert, F. Le Bolzer, and F. Schnitzler, "Flexible recurrent neural networks," in *Machine Learning and Knowledge Discovery in Databases*, 2020.

[78] C. Hu, J. Jiang, and Z. Wang, "Decentralized federated learning: a segmented gossip approach," *arXiv preprint arXiv:1908.07782*, 2019.

[79] S. Savazzi, M. Nicoli, and V. Rampa, "Federated learning with cooperating devices: A consensus approach for massive iot networks," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4641–4654, 2020.

[80] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *Journal of Parallel and Distributed Computing*, vol. 148, pp. 109–124, 2021.

[81] G. Danner and M. Jelasity, "Token account algorithms: the best of the proactive and reactive worlds," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 885–895, IEEE, 2018.

[82] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *International Conference on Machine Learning*, pp. 3478–3487, PMLR, 2019.

[83] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," tech. rep., Stanford InfoLab, 1999.

[84] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *Sixth international conference on data mining (ICDM'06)*, pp. 613–622, IEEE, 2006.

[85] M. T. Barlow, *Random walks and heat kernels on graphs*, vol. 438. Cambridge University Press, 2017.

[86] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[87] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[88] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 338–348, 2020.

[89] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *International Conference on Machine Learning*, pp. 1725–1735, PMLR, 2020.

[90] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson, "Combining label propagation and simple models out-performs graph neural networks," *arXiv preprint arXiv:2010.13993*, 2020.

[91] H. Dong, J. Chen, F. Feng, X. He, S. Bi, Z. Ding, and P. Cui, "On the equivalence of decoupled graph convolution network and label propagation," in *Proceedings of the Web Conference 2021*, pp. 3651–3662, 2021.

[92] E. Krasanakis, S. Papadopoulos, and I. Kompatsiaris, "p2pgnn: A decentralized graph neural network for node classification in peer-to-peer networks," *arXiv preprint arXiv:2111.14837*, 2021.

[93] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[94] G. Berry, A. Sirianni, I. Weber, J. An, and M. Macy, "Going beyond accuracy: estimating homophily in social networks using predictions," *arXiv preprint arXiv:2001.11171*, 2020.

[95] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[96] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, *et al.*, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[97] G. Namata, B. London, L. Getoor, B. Huang, and U. EDU, "Query-driven active surveying for collective classification," in *10th International Workshop on Mining and Learning with Graphs*, vol. 8, 2012.

[98] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[99] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[100] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[101] S. Bock, J. Goppold, and M. Weiß, "An improvement of the convergence proof of the adam-optimizer," *arXiv preprint arXiv:1804.10587*, 2018.

[102] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[103] E. Krasanakis, S. Papadopoulos, I. Kompatsiaris, and A. Symeonidis, "pygrank: A python package for graph node ranking," *arXiv preprint arXiv:2110.09274*, 2021.

[104] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[105] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi, and E. Weinan, "Towards theoretically understanding why sgd generalizes better than adam in deep learning," *ArXiv*, vol. abs/2010.05627, 2020.

[106] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[107] X. Peng, L. Li, and F.-Y. Wang, "Accelerating minibatch stochastic gradient descent using typicality sampling," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, pp. 4649–4659, 2020.

[108] L. van der Maaten and G. E. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[109] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[110] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[111] H. Fassold, "Some like it tough: Improving model generalization via progressively increasing the training difficulty," *CoRR*, vol. abs/2110.13058, 2021.

[112] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations*, 2018.

[113] A. Ruiz and J. Verbeek, "Adaptative inference cost with convolutional neural mixture models," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1872–1881, 2019.

[114] A. Ruiz and J. Verbeek, "Distilled hierarchical neural ensembles with adaptive inference cost," *arXiv preprint arXiv:2003.01474*, 2020.

[115] S.-I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," *arXiv preprint arXiv:1902.03393*, 2019.

[116] T. Mayet, A. Lambert, P. Le Guyadec, F. Le Bolzer, and F. Schnitzler, "SkipW: Resource adaptable RNN with strict upper computational limit," in *International Conference on Learning Representations*, 2020.

[117] A. Ozerov and N. Q. K. Duong, "Inplace knowledge distillation with teacher assistant for improved training of flexible deep neural networks," in *29th European Signal Processing Conference, EUSIPCO 2021*, (Dublin, Ireland), Aug. 2021.

[118] A. Pérez-Salinas, D. López-Núñez, A. García-Sáez, P. Forn-Díaz, and J. I. Latorre, "One qubit as a universal approximant," *Phys. Rev. A*, vol. 104, p. 012405, Jul 2021.

[119] J. S. Garofolo, "Timit acoustic phonetic continuous speech corpus," *Linguistic Data Consortium, 1993*, 1993.