Good Practices on Parallel and Distributed Programming for Training Neural Networks

John Anderson García Henao

ARTORG Center for Biomedical Engineering Research

University of Bern

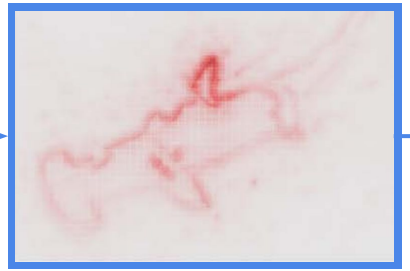
UNIVERSITÄT BERN

The remarkable progress of deep learning in CV and NLP

Input

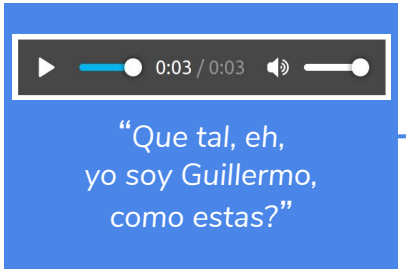
Deep Representation

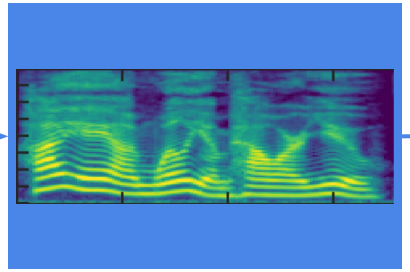
Output



"Hammerhead shark"

Deep Taylor (GoogleNet)
https://arxiv.org/pdf/1512.02479v1.pdf





"How's it going, hey,
this is Guillermo, How
are you?"

Mel spectrograms (WaveRNN)
https://arxiv.org/pdf/1904.06037.pdf

Distributed Deep Learning in Healthcare

2016 - Deep Learning for Detection of Diabetic Eye Disease, by Google Brain Team

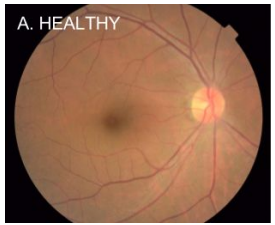
JAMA doi:10.1001/jama.2016.17216

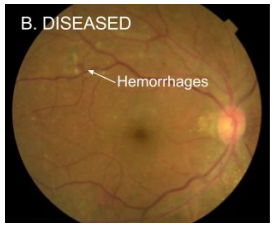
Diabetic retinopathy is the fastest growing cause of blindness

+ By 2016 there were more than **415 million diabetic patients at risk.**

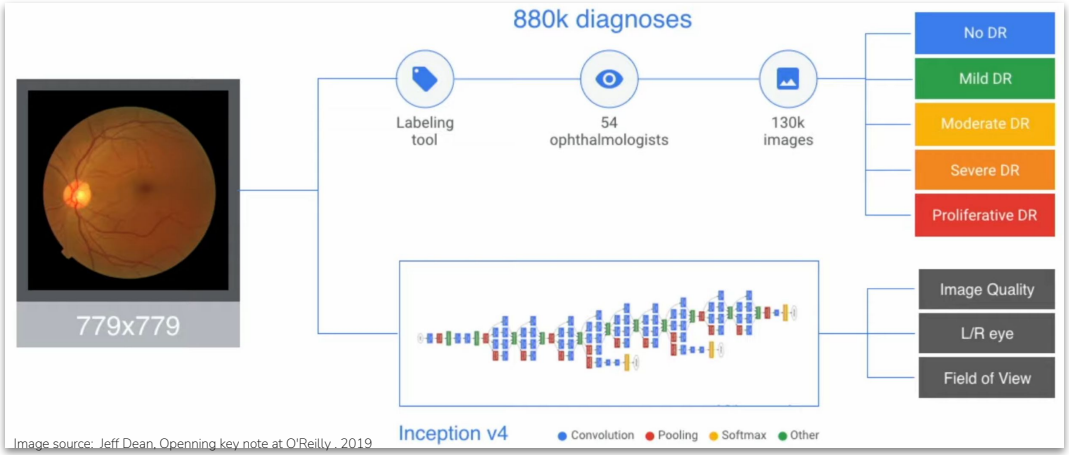
+ If caught early, the disease can be treated; if not, it can lead to irreversible blindness.“

+ The severity task is determined by *by the type of lesions present, like hemorrhages, microaneurysms, hard exudates and others.*





Deep Learning Workflow



+ They applied a distributed stochastic gradient descent implementation to train the network weights, using model replication asynchronously over a cluster of nodes.

** Large Scale Distributed Deep Networks, https://dl.acm.org/doi/10.5555/2999134.2999271

“Developing neural network image classification models often requires significant architecture engineering”

Automatic Machine Learning For Distributed Systems

2018 - Learning Transferable Architectures for Scalable Image Recognition, by Google Brain Team

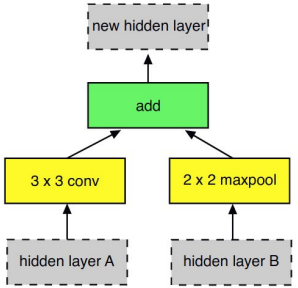
https://arxiv.org/pdf/1905.11946.pdf

Neural Architecture Search called NASNet search space

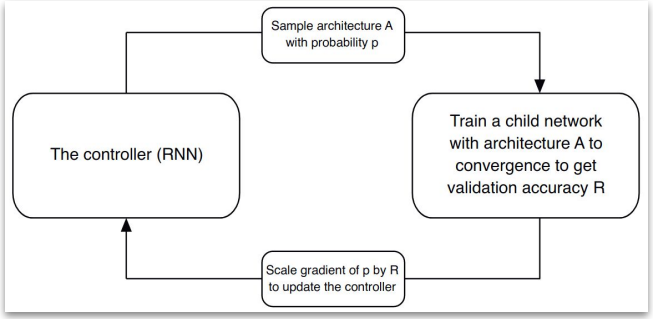
+ The main feature is search space enable transferability:

*NASNet search the best **convolutional cell** on a small dataset (CIFAR-10) and then applied to the larger dataset (ImageNet).*

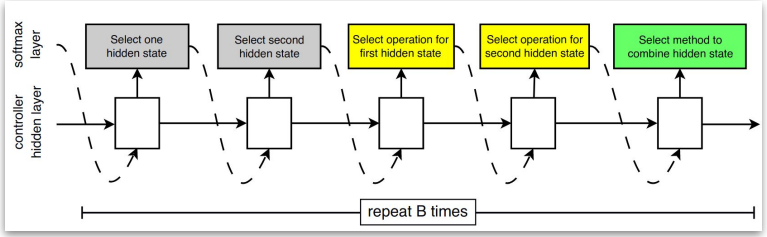
*** CIFAR-10:** *60,000 images in 10 different classes.*
*** ImageNet: >** *14 million images with more than 20.000 classes.*



Convolutional Cell



NASNet General Schema



The RNN Controller model architecture

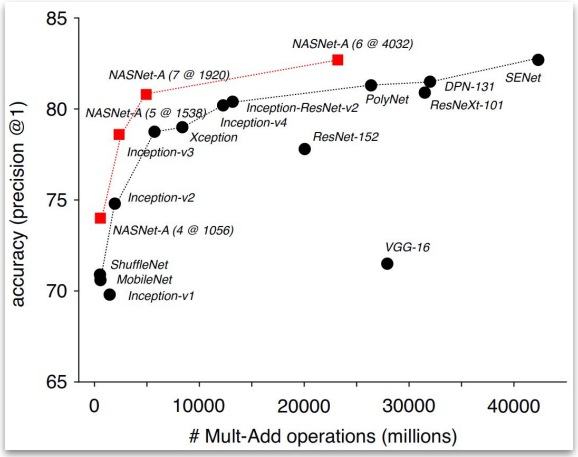
Automatic Machine Learning For Distributed Systems

2018 - Learning Transferable Architectures for Scalable Image Recognition, by Google Brain Team.

https://arxiv.org/pdf/1905.11946.pdf

Neural Architecture Search called NASNet search space

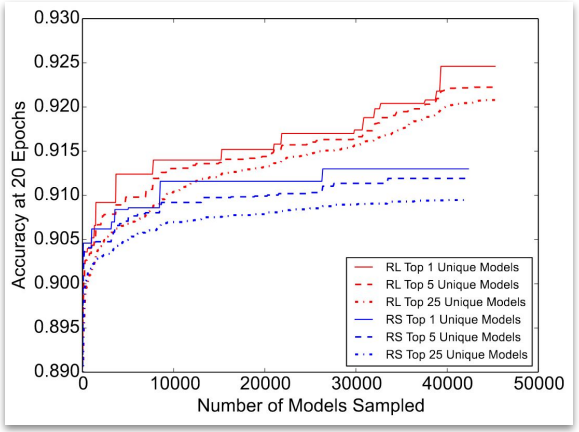
+ NASNet found a model constructed from the best cell achieving 1.2% better in top-1 accuracy than the best human-invented architecture.



Results over the ImageNet 2012 challenges

Training the NASNet using a distributed approach:

+ The controller generate a distributed worker pool with 450 tasks.
+ For that pool, they used 450 GPUs concurrently at any time.



Efficiency comparison of random search versus reinforcement learning for NASNet.

Distributed Deep Learning in Healthcare

2019 - Exascale Deep Learning to Accelerate Cancer Research, by Oak Ridge National Lab.

IEEE doi:10.1109/BigData47090.2019.9006467

Multi-objective optimization for deep learning convergence and effectively utilizing HPC systems. (MENNDL)

*** MENNDL:** *Multi-node Evolutionary Neural Networks for Deep Learning.*

Quantification task: Tumor Infiltrating Lymphocytes (TILs)

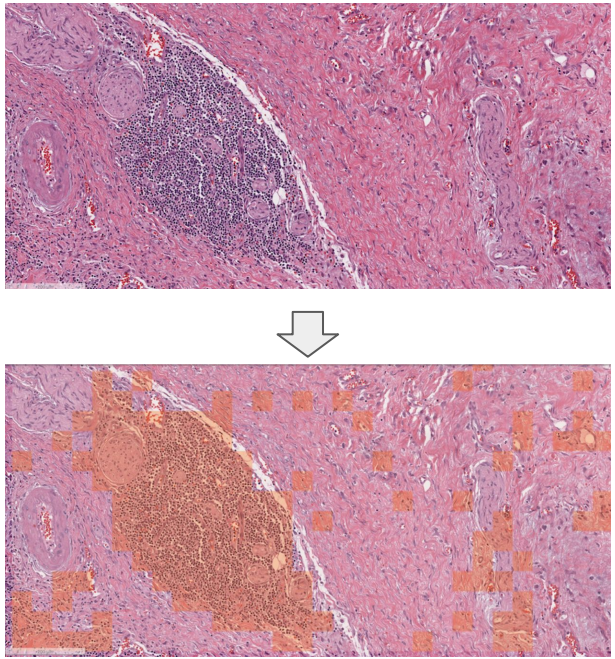
"During the cancer diagnosis and treatment process, a patient may have a biopsy, which produces a diagnostic tissue sample."

+ From whole slide image they extract patches of 50x50 um, and each one could contains from 100.000 to 1000.000 cell nuclei.

+ The dataset used has 86,000 patches (with 20% TIls Positives) and all the patches has an image with 100x100 pixel resolution.

+ It contains seven different cancer types in the breast, colon, lung, pancreas, prostate, skin, and pelvic forms.

Source slide image



Quantization Result

Distributed Deep Learning in Healthcare

2019 - Exascale Deep Learning to Accelerate Cancer Research, by Oak Ridge National Lab.

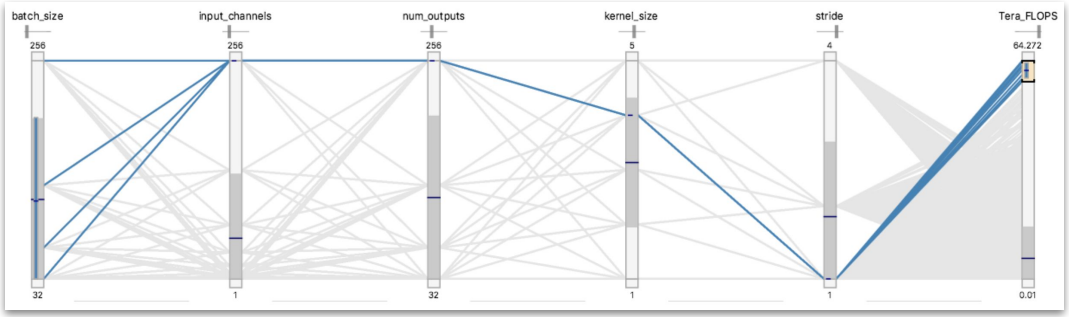
IEEE doi:10.1109/BigData47090.2019.9006467

We can generate a single model that can study the seven different types of cancer?

Evolutionary multi-objective optimization

+ The objective function is compose by a vector-valued like:
  (eg., speed or accuracy).

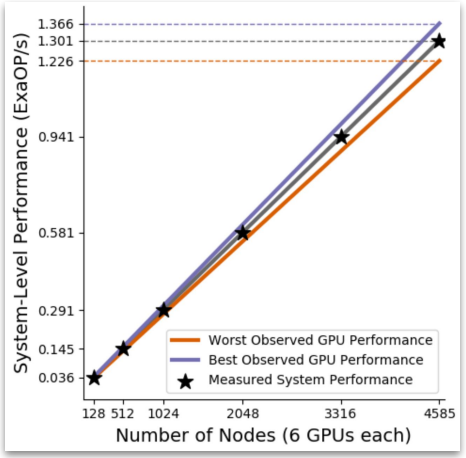
+ MENNDL, starts with some initial set of hyperparameters of ***Inception Neural Architecture.***



Parallel hyperparameter search for 10.000 convolution layers using tensor.

4,608 Nodes, with a total of:
9,216 CPUs,
27,648 GPUs and
10 PB DDR4

Each Node, has:
2 IBM Power9 CPUs,
6 Nvidia Volta GPUs and
512 GB DDR4



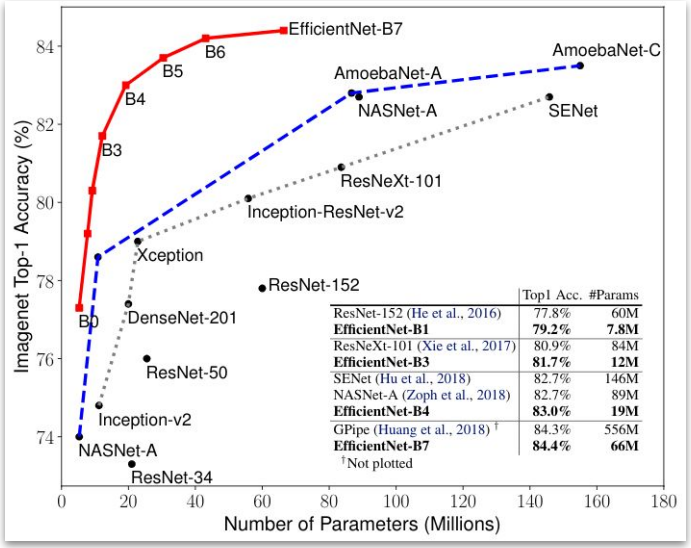
Weak scaling results

Automatic Machine Learning For Distributed Systems

2020 - EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Google Brain Team.

https://arxiv.org/pdf/1905.11946.pdf

Multi-objective function for model scaling.

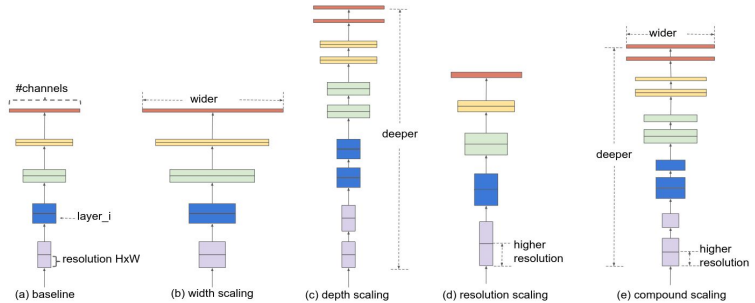


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Results over the ImageNet 2012 challenges

+ There is a principles method for scaling CNNs that can achieve better accuracy and efficiency?



+ The optimization goal:

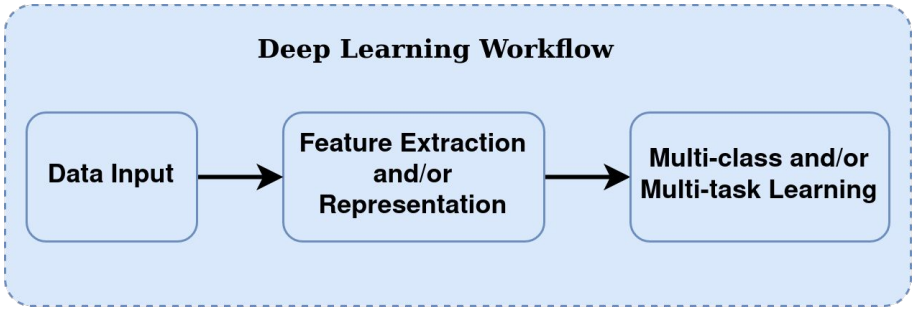
$$ACC(m) \times [FLOPS(\bar{m})/T]^{w}$$

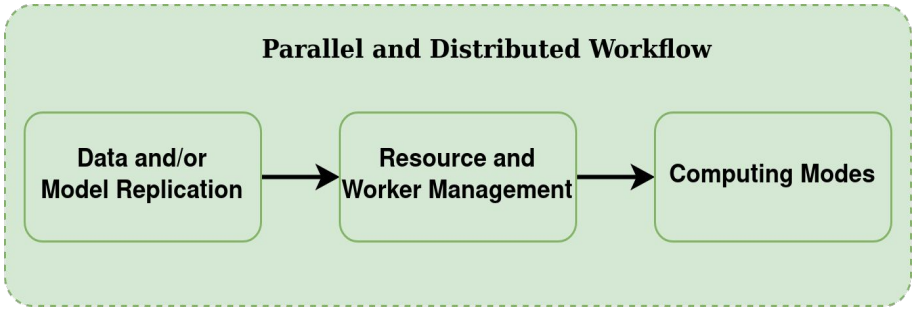
Where,

m denotes the model;
T is the target FLOPS;
W=-0.07 is a hyperparameter for controlling the trade-off.

What are the main components to develop a distributed deep learning application?

Main components to develop a distributed deep learning application





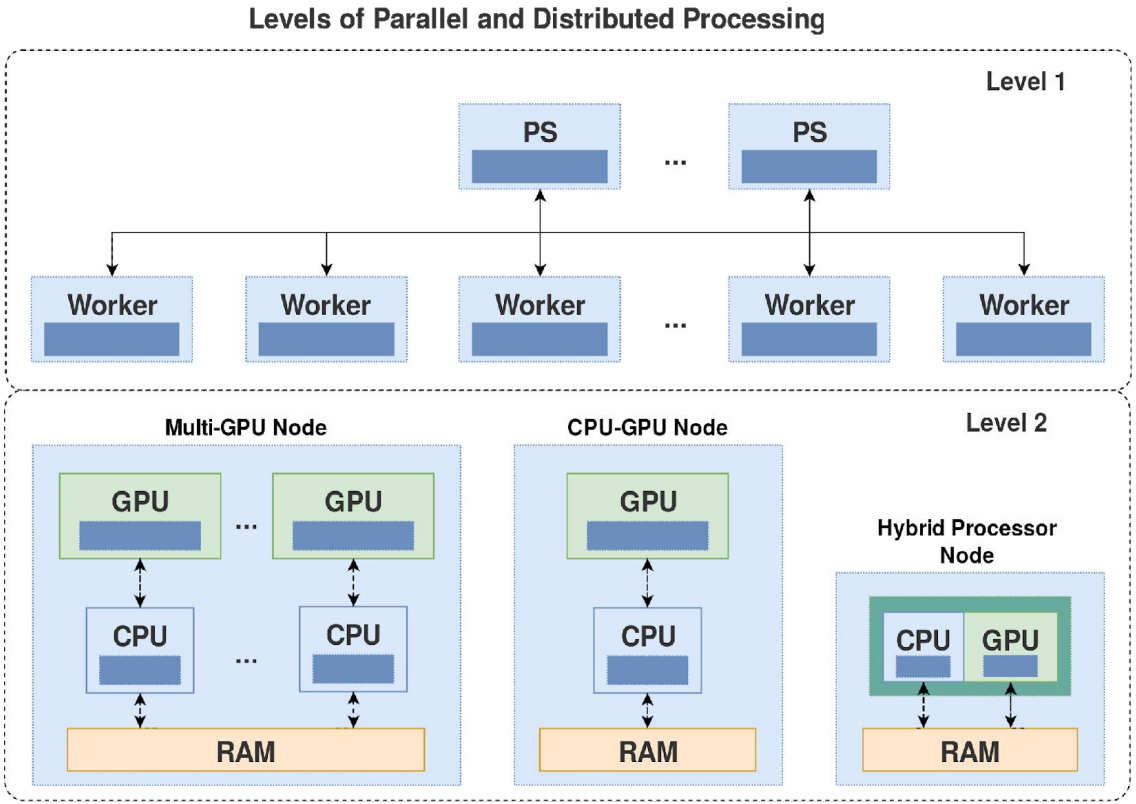
Performance metrics:

- Accuracy
- Generalization
- Interpretability
- Robustness

Performance metrics:

- Execution Time
- Portability
- Scalability
- Energy Efficiency

Computation modes to consider for training a neural network



In a cluster of nodes:

- *GRPC*
- *MPI*

In a multi-GPU server:

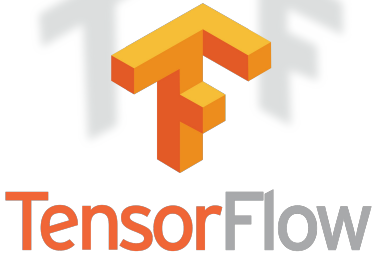
- *Synchronous*
- *Asynchronous*

In a desktop computer:

- *Exploit the memory*
- *Batch execution*

Main Optimized Frameworks to Accelerate Deep Learning Training on GPUs

Apache Software Foundation

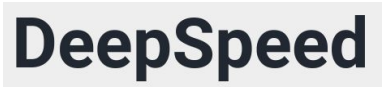


Google Brain Team

Facebook's AI Research lab

Uber Engineering Team

Built on top of TensorFlow 2.0



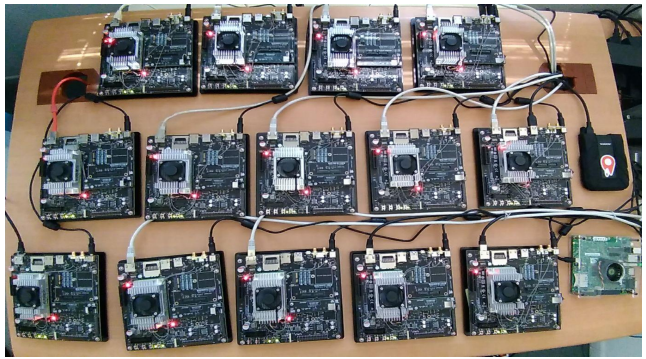
https://arxiv.org/abs/1910.02054

https://hal.archives-ouvertes.fr/hal-02869960

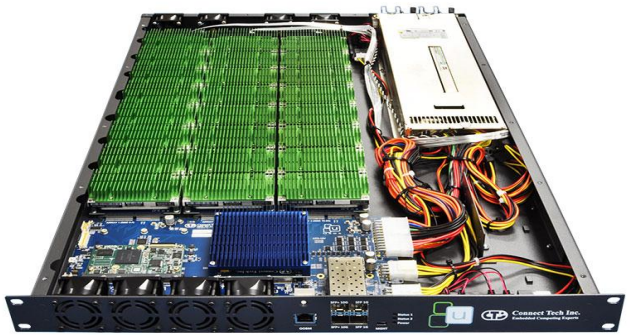
https://arxiv.org/pdf/2007.13518

DiagnoseNET

Repository: https://github.com/IADBproject/diagnosenet

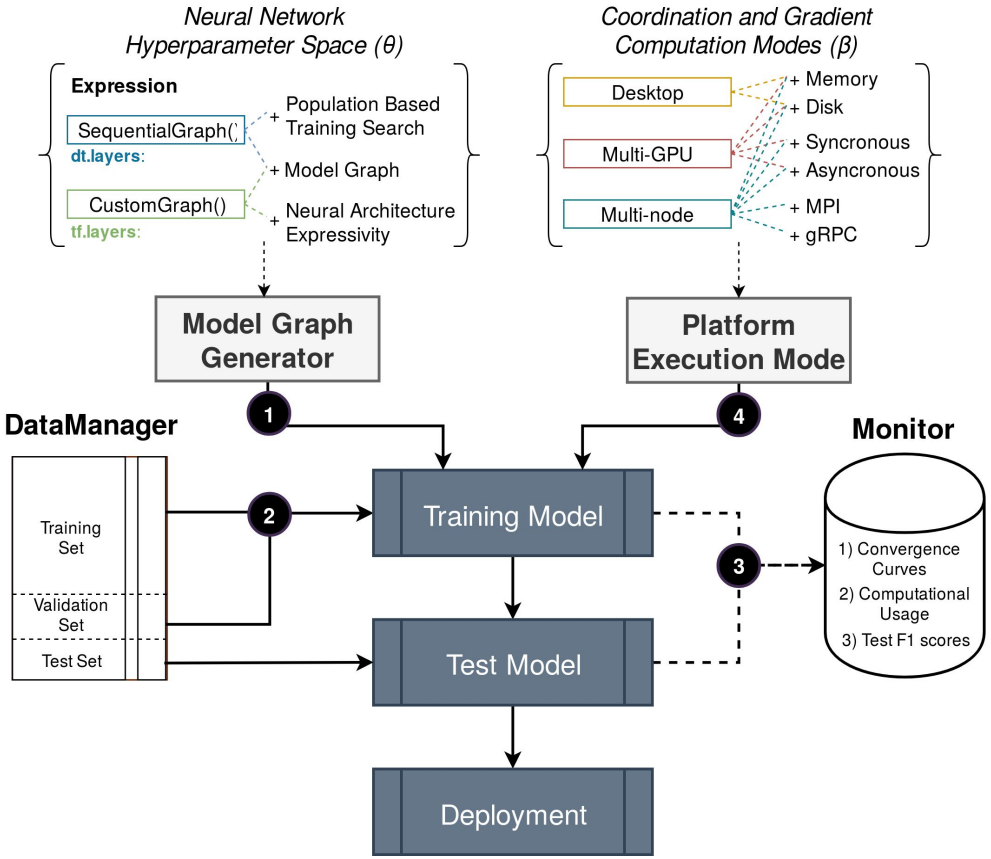


Mini-Cluster Jetson TX2

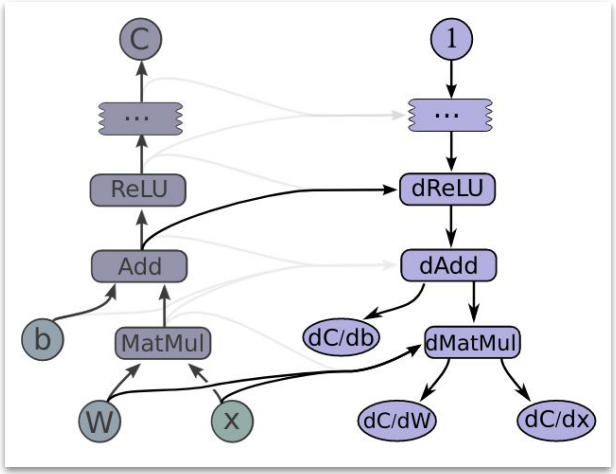


Array Node with 24 Jetson TX2

DiagnoseNET: Programming Framework Scheme

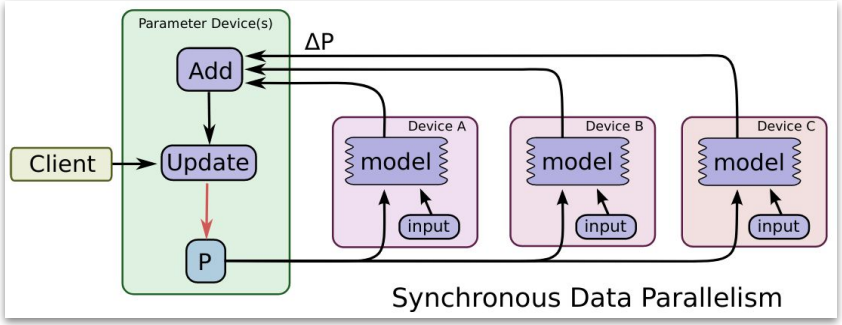


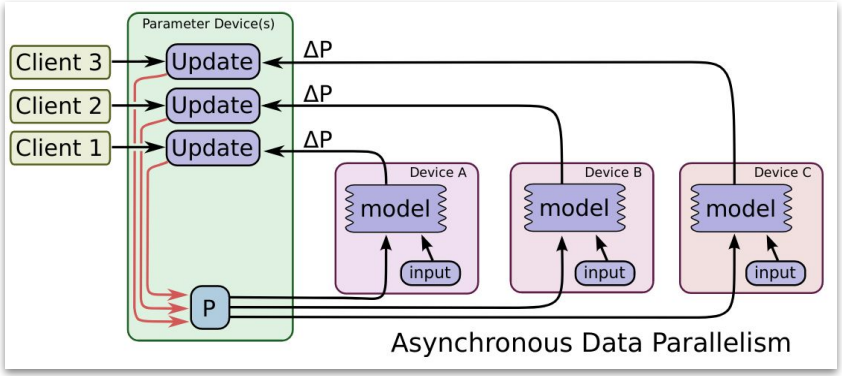
Synchronous and Asynchronous Data Parallel Training



Example of Gradients computed for a graph in TensorFlow

https://arxiv.org/pdf/1603.04467.pdf





DiagnoseNET Model Graph Generator and Data Manager

Step 1: Model definition to generate several graphic-model objects.

```
import diagnosenet as dt
stacked_layer_1 =  [dt.Relu(14637, 2048),
                    dt.Relu(2048, 1024),
                    dt.Relu(1024, 1024),
                    dt.Linear(1024, 14)]

model_1 = dt.sequentialGraph(
                input_size=14637, output_size=14,
                layers=stacked_layer_1,
                loss=dt.CrossEntropy,
                optimizer=dt.Adam(lr=0.001))
```

Step 2: Dataset splitting and micro-batching over the workers.

```
data_config_1 = dt.Batching(
                dataset_name="medical_D1", valid_size=0.05,
                devices_number=4, batch_size=128)
```

DiagnoseNET for Distributed Training with gRPC

Step 3: Select the computing modes as gRPC asynchronous replication

```
platform = dt.Distibuted_GRPC(
        model=model_1,
        datamanager=data_config_1,
        monitor=enerGyPU(machine_type="arm"),
        max_epochs=20,
        ip_ps=argv[0], ip_workers=argv[1])

platform.asynchronous_training(
        dataset_path=/myworkspace/datasetpath,
        inputs_name="X.npy", targets_name="Y.npy",
        job_name=argv[0], task_index=argv[1])
```

Step 4: Distributed orchestration with GRPC asynchronous.

```
import diagnosenet as dt
dt.between_graph_replication(
        d_replica_path=/myworkspace,
        d_replica_name="GRPC_replica.py",
        ip_ps="host1",
        ip_workers="host2,host3,host4,host5",
        num_ps=1, num_workers=4)
```

DiagnoseNET MPI Synchronous and Asynchronous Algorithms

Algorithm 1 Synchronous MPI Kernel

```
if master True then
    masterInput ← {dataset, workers}
    DistributedBatching(dataset, workers)
else
    workerInput ← {batches, hyperparameters}
    model ← sequentialGraph(hyperparameters)
while ConvergenceCondition do
    if master True then
        for all worker ∈ workers do
            masterGrads ← received(workerGrads)
        averageGrads ← average(masterGrads)
        send(averageGrads)
    else
        workerGrads ← compute(model, batches)
        send(workerGrads)
    if master True then
        for all worker ∈ workers do
            masterLoss ← received(workerLoss)
        averageLoss ← average(masterLoss)
        if overfitting(averageLoss) True then
            send(averageLoss, earlyStopping)
        else
            send(averageLoss, False)
    else
        workerWeights ← received(masterWeights)
        projection ← model.Apply(workerWeights)
        workerLoss ← computeLoss(projection, labels)
        send(workerLoss)
```

Algorithm 2 Asynchronous MPI Kernel

```
if master True then
    masterInput ← {dataset, workers}
    DistributedBatching(dataset, workers)
else
    workerInput ← {batches, hyperparameters}
    model ← sequentialGraph(hyperparameters)
while ConvergenceCondition do
    if master True then
        convergeFlag ← received(workerCond)
        masterGrads ← received(workerGrads)
        collectGrads ← collection(masterGrads)
        averageGrads ← average(collectGrads)
        send(averageGrads)
    else
        if overfitting(averageLoss) True then
            send(averageLoss, earlyStopping)
        else
            send(averageLoss, False)
        if decrease(averageLoss) True then
            send(Updated(masterWeights))
        workerGrads ← compute(model, workerInput)
        send(workerGrads)
    if master False then
        workerWeights ← received(masterWeights)
        projection ← model.Apply(workerWeights)
        workerLoss ← computeLoss(projection, labels)
```

DiagnoseNET Model Graph Generator and Data Manager

Step 1: Model definition to generate several graphic-model objects.

```
import diagnosenet as dt
stacked_layer_1 =  [dt.Relu(14637, 2048),
                    dt.Relu(2048, 1024),
                    dt.Relu(1024, 1024),
                    dt.Linear(1024, 14)]

model_1 = dt.sequentialGraph(
                input_size=14637, output_size=14,
                layers=stacked_layer_1,
                loss=dt.CrossEntropy,
                optimizer=dt.Adam(lr=0.001))
```

Step 2: Dataset splitting and micro-batching over the workers.

```
data_config_1 = dt.Batching(
                dataset_name="medical_D1", valid_size=0.05,
                devices_number=4, batch_size=128)
```

DiagnoseNET for Distributed Training with MPI

Step 3: MPI asynchronous platform execution modes

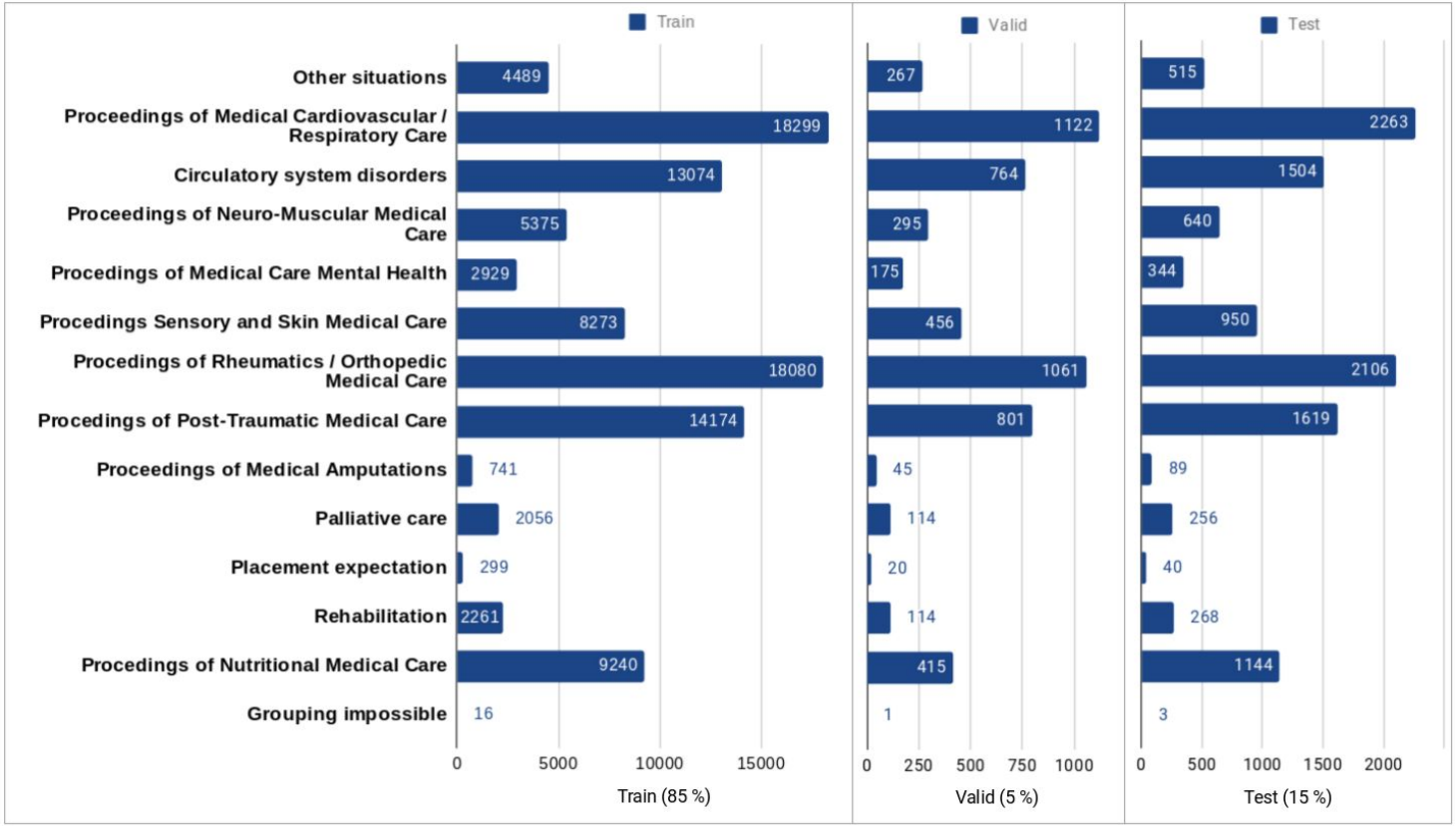
```
platform = dt.Distibuted_MPI(
        model=model_1,
        datamanager=data_config_1,
        monitor=enerGyPU(machine_type="arm"),
        max_epochs=20, early_stopping=3])

platform.asynchronous_training(
        dataset_name="medical_D1",
        dataset_path=/myworkspace/datasetpath,
        inputs_name="X.npy", targets_name="Y.npy",
        weighting=1)
```

Step 4: Launcher the training process using the MPI options

```
mpirun -np 5 --hostfile <filename> python3.6 mpi_processing.py
```

Case of Study 1: Medical Care Purpose Classification for In-Patients



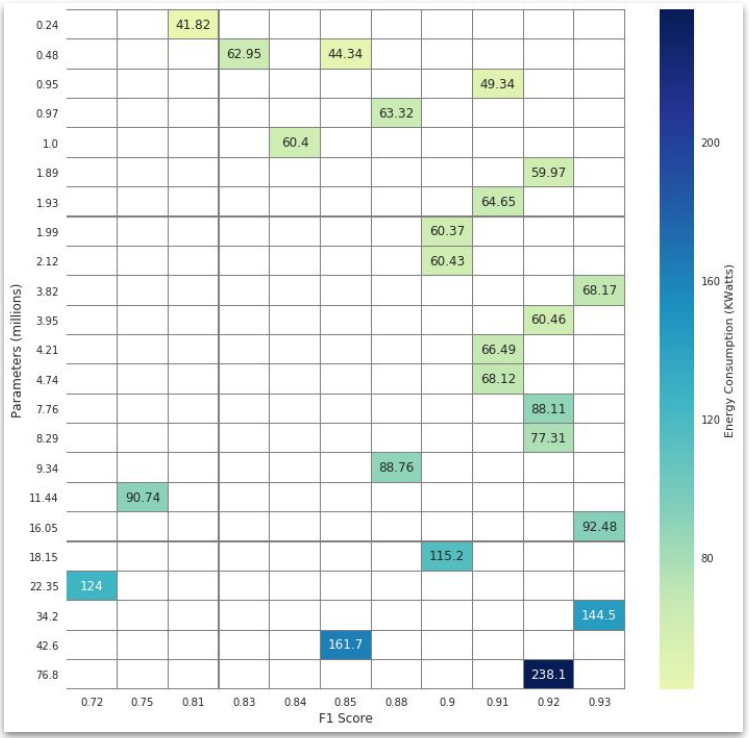
+ In-Patients Distribution for the PACA Clinical Dataset (2008: 121.369 In-Patients)

MLP Hyperparameter Search to Classify The Care Purpose

Search Space Model Descriptors.

Hyperparameters (d)	Hyper. Configurations (n)	State
Learning rate	0.0005, 0.001, 0.005, 0.01, 0.05, 0.1	Fixed: 0.001
Activation function	relu, tanh, linear	Fixed: relu
Num. Units per layer	16, 32, 64, 128, 256, 512, 1024, 2048, 4096	Search
Num. hidden layers	2, 4, 8, 16	Search
Regularization	Dropout: 0.6, 0.7, 0.8	Fixed: 0.8
Batch size	24.576, 12.288, 6.144, 3.072, 1.536, 768	Search
Num. of workers	4, 6, 8, 10, 12	Search

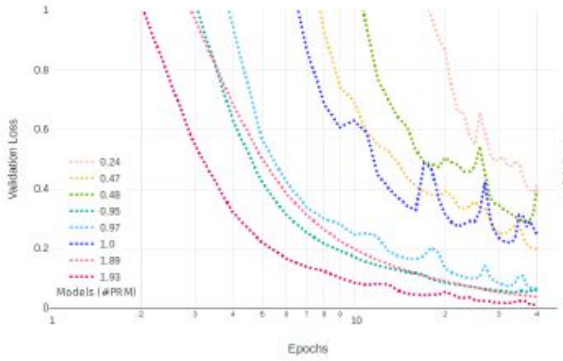
Accuracy vs Energy Consumption.



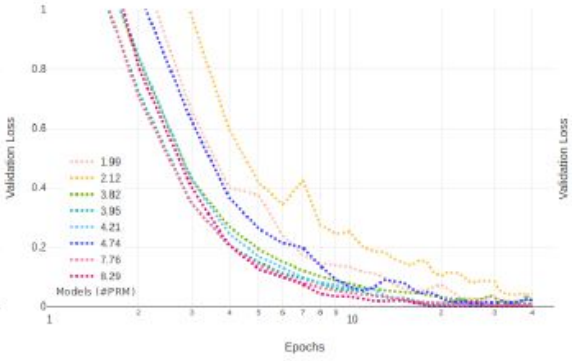
Model Dimension Space in Number of Parameters

MLP Hyperparameter Search to Classify The Medical Task 1

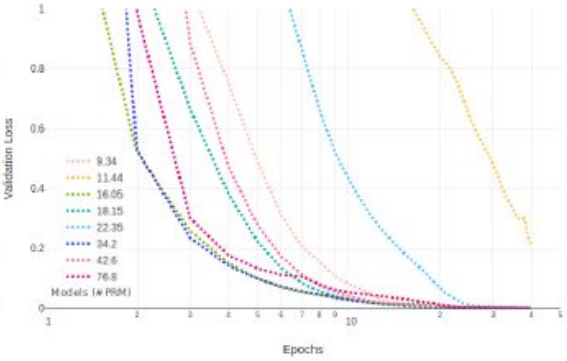
(a) Fine-grain convergence validation.



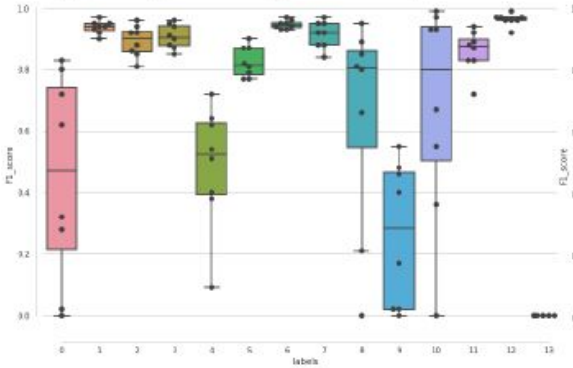
(b) Middle-grain convergence validation.



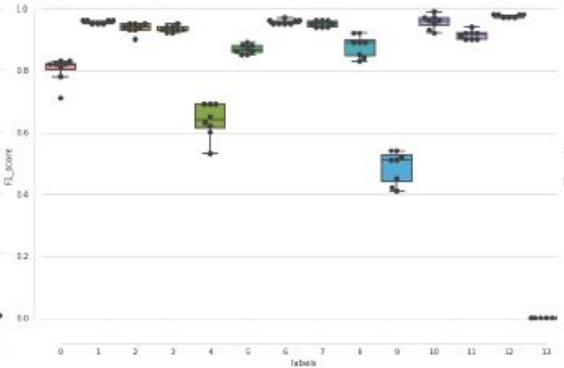
(c) Course-grain convergence validation.



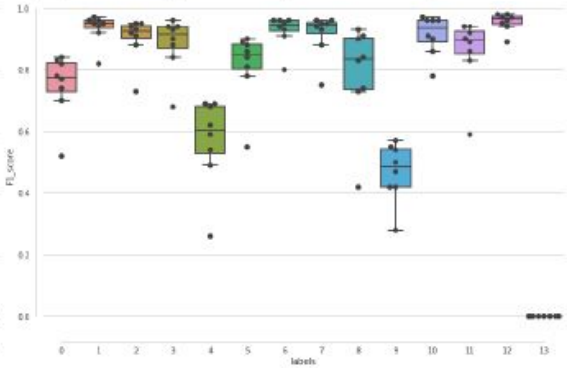
(d) Fine-grain test prediction by class.



(e) Middle-grain test prediction by class.



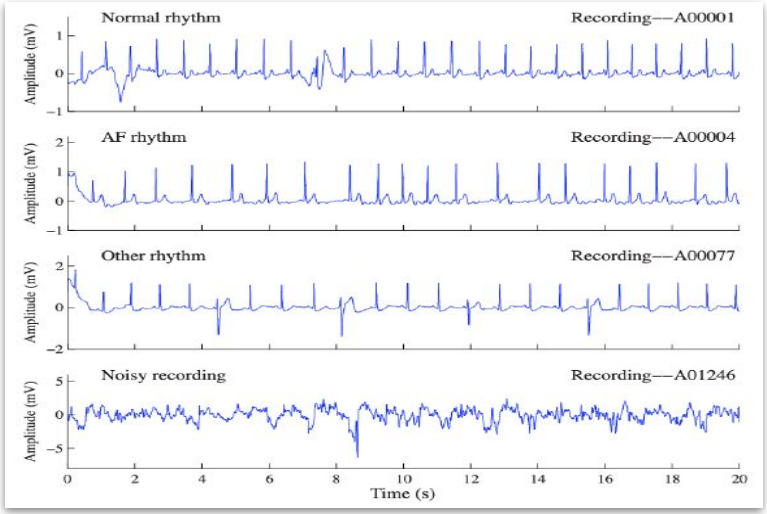
(f) Course-grain test prediction by class.



Case of Study 2: Atrial Fibrillation Classification for Cardiac Diagnosis

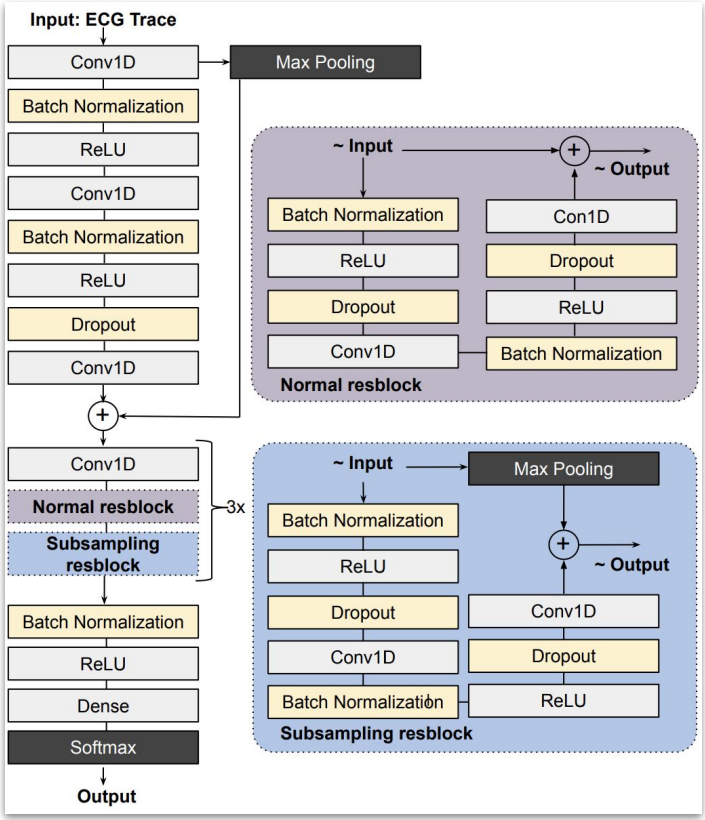
Distribution of Cardiac Arrhythmia Labels

Class	Label Description	Source dataset	Training Dataset	Small Samples
0	Normal	5,050	34,303	4241
1	AF	738	6,542	815
2	Others	2,456	18,986	2424
3	Noisy	284	1,382	171

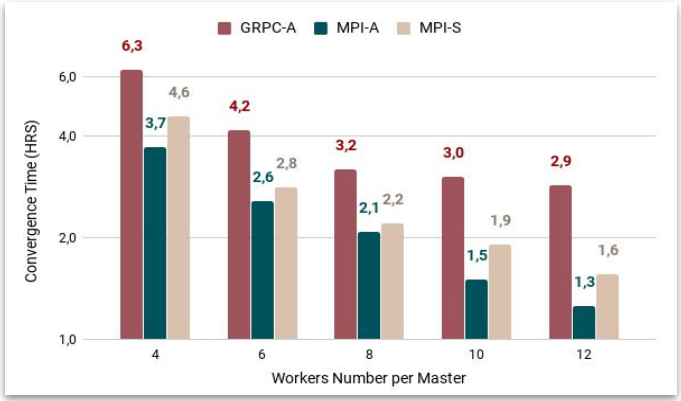


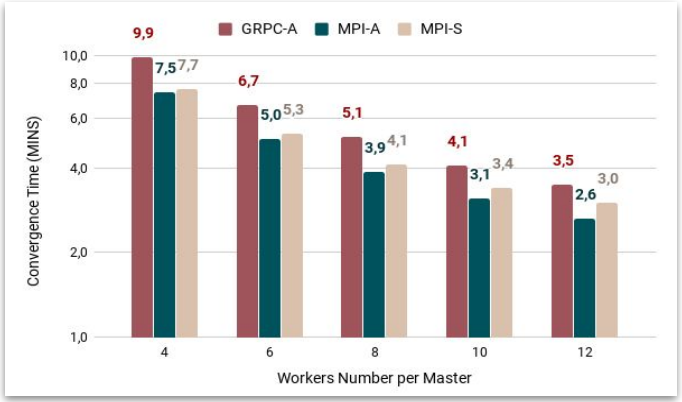
Examples of the ECG waveforms

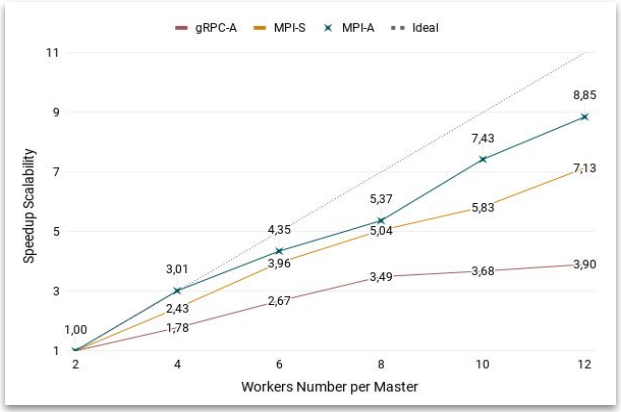
ECG Convolutional Neural Network

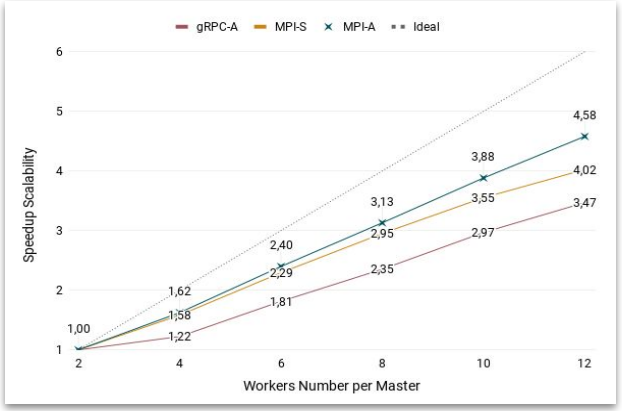


Worker Scalability for Training the Two Case of Studies









Medical Care Purpose Classification

Atrial Fibrillation Classification

Future Works and Acknowledgements

1) Add population search methods for model generalization and scalability training of neural networks on HPC systems.

2) Add a dedicated module for federation learning experiments with asynchronous distributed learning and contribute to data autonomy in healthcare centers.

+ We thank Michel Riveill, Frédéric Precioso and Pascal Staccini for its orientation and support to develop the ***DianoseNet project.***

+ We thank Felix Mejia, Ziqing Du, Mohamed Younes, Arno Gobbin and Chap Chanpiseth for its develop contributions.

+ We thank, Carlos Jaime, Luis Núñez, Thomas Fryer and Mario Reale for the support to continue with the development of the ***DiagnoseNET project.***

Gracias por tu atención!

Preguntas e inquietudes: jagh1729@gmail.com

John Anderson García Henao

ARTORG Center for Biomedical Engineering Research

University of Bern

UNIVERSITÄT BERN