KU LEUVEN

# Learning to learn: automatic design of neural network architectures for biomedical signal processing tasks

Aymeric Gaspard Barbin

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

**Thesis supervisors:**
Prof. dr. ir. Alexander Betrand
Prof. dr. ir. Maarten De Vos

**Assessor:**
Prof. dr. ir. Peter Karsmakers

**Mentor:**
Ir. Nick Seeuws

Academic year 2020 – 2021

# Preface

This thesis is the result of personal work, but it would not have been possible without the involvement of the following people who I would like to thank.

First, I would like to express my gratitude to the supervisors for allowing me to take a first step in a research project, and to work on such an exciting subject.

I would like to thank the technicians of the VSC for helping me many times and with whom I tried to make the most of the possibilities offered by this environment. I must also acknowledge Johan Suykens and his student Bram De Cooman for their technical advices regarding optimization in Reinforcement Learning.

Finally, I would like to salute my mentor Nick for his availability, his encouragement and his practical advice which allowed me to better advance in my work.

*Aymeric Gaspard Barbin*

# Contents

# Abstract

The impact of Machine Learning in all areas of innovation is increasingly growing and this is particularly the case in the field of medical research where the statistical signatures linked to a particular biological event, like the electrical cortical activity, can now be captured and analysed in an automated way.

This thesis aims at exploiting the power of Neural Architecture Search (NAS) to automate the search of adequate neural networks for classifying electro-corticographic outputs. A NAS algorithm was developed on a specific case study of electro-encephalographic classification, i.e, sleep-stage classification from preterm infant electroencephalograms (EEG).

Although the utility of NAS algorithms is not to be proven anymore, it is not always that the sampled architectures outperform manually designed networks, notably because of the necessary high computing power and time involved to conduct the search, that is why it was necessary to make the best use of the latest advances in the field. The NAS was designed to explore for a broad search space and take different kind of inputs (not only EEG). The results of this work are threefold. Firstly, a NAS algorithm based on state-of-the-art features was implemented, leading to major improvements in term of speed and end results, a new version of the reward function leading to more scalability and faster convergence is also introduced. Secondly, a sound methodology to use the NAS and ensure an increasing improvement of the results is given, its efficiency is notably due to its scalability in term of testing, and analysis, based on small experiments. Thirdly, the reliability of the methodology is shown, making the NAS controller sample increasingly better architectures and leading to the discovery of new design principles on the case of study. Finally, improvements suggestions and ideas for future research are discussed.

# List of Figures and Tables

## List of Figures

## List of Tables

# List of Abbreviations and Symbols

## Abbreviations

| | |
|---|---|
| NAS | Neural Architecture Search |
| DAG | Directed Acyclic Graph |
| PSNR | Peak Signal-to-Noise ratio |
| Conv(2,5) | Conv layer with filters of height 2 and width 5 |

## Symbols

| | |
|---|---|
| $\nabla$ | Gradient |

# Chapter 1

# Introduction

The main purpose of the thesis was to investigate the possibilities of Neural Architecture Search (NAS) in order to develop enhanced neural networks for electroencephalogram (EEG) classification. In this introductory chapter, a background of the field of EEG and the context of this work is given as well as a brief outline of the overall thesis structure.

## 1.1 Background

Sleep is a critical component for the development of infants as it enables the development and the maturation of their neurosensory system.

Sleeping cycles start to appear before birth - around 28 weeks of postmenstrual age (PMA) [1] - and consist of two stages, rapid eye movement sleep (REM) and quiet sleep (QS or non-REM). Theses two stages are of prime importance for the development of the hippocampal system (memory creation), limbic system (emotional experience), visual system, auditory system, and inferior parts of the brain such as pons thalamus and brain stem centers [8]. Overall, their development directly affects the infant brain plasticity and its ability of memorizing. For term children, sleep cycles generally regulate naturally, and can be identified into four distinct stages that appear just before birth. On the contrary, for preterm children (born before 37 weeks PMA), the sleep cycles are less stabilized and consist first of two stages only, active stage (AS) and QS. Therefore, is it very important to monitor sleep stages of preterm babies in order to provide the necessary nursing care that help regularizing their sleep.

In order to detect sleep-wake states in infants, a diagnosis based on the observation of biobehavioral responses is usually performed, at first, by nurses, then, a manual study of the EEG by an expert physician generally gives a definitive insight. This process being time consuming, several automating methods have been developed to reduce the workload in intensive care units (NICU).

A first approach of the automation process was to assess EEG according to several characteristics such as frequency content [16], proportional duration of bursts

[1] number of weeks elapsed since the first day of the last menstrual cycle of the mother

[14] or time profiles [5]. But theses characteristics are the result of manual human research and do not account for all potential hidden features of EEG signals. Because of a low signal-to-noise ratio, a lot of relevant features are hidden to the human eye. This is where Deep Learning comes into play as it is well-known for its ability to detect underlying features on non-stationary and nonlinear data. Convolutional neural networks (CNN) have achieved good performances in the domain of EEG classification, in particular in the work of Ansari et al. [2], where a CNN was developed that outperformed existing algorithms for EEG-based classification of sleep staging for preterm and term infants.

## 1.2 Goal

This thesis aims at investigating the possibilities offered by the field of Neural Architecture Search in order to explore enhanced architectures of CNN in the domain of EEG classification. Indeed, manual neural architecture design is a subtle art that requires knowledge in the domain of application as well as in Machine Learning and statistics, in the case of CNN, the complexity involved can make the design process overwhelming, and researchers often spend more time implementing their ideas than focusing on the ideas themselves. This is where NAS can help lighten the implementation part and give the researcher more time to focus on higher level aspects, such as the design of the search space.

As the range of possibilities regarding NAS development and possible search spaces for the CNN architectures is very wide, the focus of the thesis was restricted to the implementation of a NAS algorithm based on Reinforcement Learning and the target architectures were limited to CNN for the classification of sleep-stage EEG and in particular of QS EEG (binary classification).

## 1.3 Thesis Overview

In chapter 2 theoritical background is given, where convolutional neural networks, which are the target architectures searched by our NAS algorithm, are introduced, then a detailed overview of NAS in general is shown, as well as the specific paradigm under which the NAS algorithm of the thesis falls, in the Search Strategy part are also explained two improvements that were made to increase the efficiency of the algorithm. Finally, a Related Work section introduces the CNN from Ansari et al. [2] which was used as a reference during all the work, and an overview of already existing EEG-classification CNN is given, for every architecture, design principles and the choices of layers are highlighted.

The data for the thesis is introduced in chapter 3 where a theoretical explanation concerning EEG is given in the preamble. In chapter 4, a self-improving method to use the NAS algorithm that ensure increasing performances is explained.

In chapter 5, the results obtained are presented and analysed. We first show the results regarding the NAS algorithm efficiency compared to other solutions, and we then present the results of the NAS itself.

Chapter 6 consists of a discussion about the results and gives suggestions for future work.

# Chapter 2

# Learning and Searching

Before going into the details of NAS, a short introduction of convolutional neural networks is necessary, as they are the target architectures searched by the NAS algorithm developed during the thesis. Best practices for CNN are also shown, as they will be used for the construction of the NAS search space.

## 2.1 Convolutional Neural Networks

Convolutional neural networks or CNN for short, were inspired by how visual information is processed in the brain. When light reaches the retina, the photons get absorbed by ganglion cells and this information flows along the optic nerves and into the brain. The first layers of neurons that process this incoming information activate for stimuli that correspond to low-level features of the input (edges, black or white colors for instance), then as the information flows, higher level features are extracted to eventually form an image with meaning.

CNN intend to treat the input information in a similar way, and use two different kind of layers to do so. The convolutional layers and the downsampling (or reduction) layers. Figure 2.1 shows an illustration of such a network [1].

Convolutional layers (also called Normal layers in the specific NAS paradigm) consist of different filtering operations that span and convolve [2] the input along its different axis to capture specific features, for example one filter could be used to capture a simple shape such as a cross on the input (see figure 2.2).

The output of this operation is called a *feature map*, and each of its elements represents the activation strength of the pattern on a particular location of the input, for example, in figure 2.2, *6* is the highest number of the feature map which confirms that the top left corner of the input is the most similar to a cross shape. Each filter creates a new feature map and theses feature maps are then stacked to form the output of a convolutional layer.

[1] image from https://fr.mathworks.com/discovery/convolutional-neural-network-matlab.html

[2] element wise multiplication between two matrices

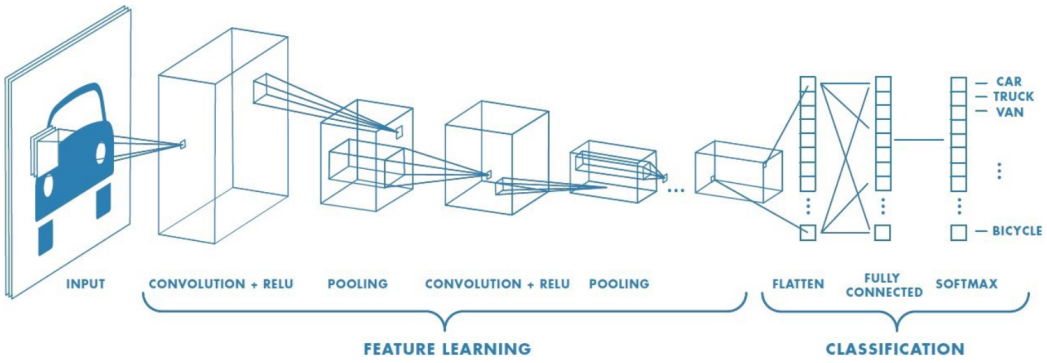



Figure 2.1: Overall structure of a CNN

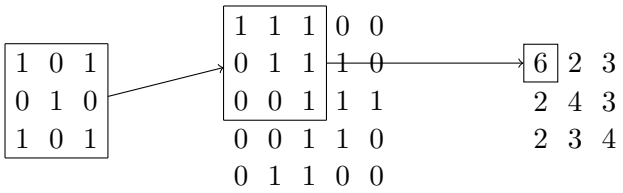



Figure 2.2: **left**: filter, **middle**: input, **right**: result

Reduction layers play the role of dimensionnality reduction, they take as input a convolutional layer output, span it and, for each spanned portion, take the maximum value ("max-pooling" layer) or an average value ("average-pooling" layer). Figure 2.3 shows the result of a max-pooling operation.

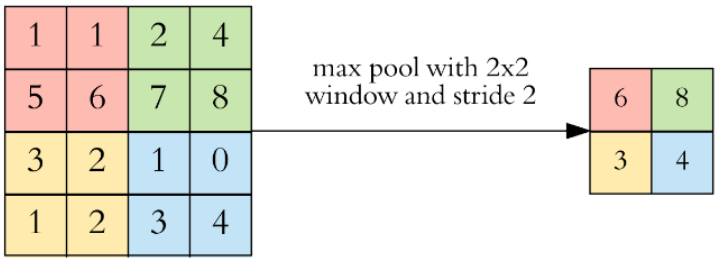



Figure 2.3: **left**: input, **right**: result

The alterning of convolutional and reduction layers constitute the *feature extraction* part of the network. This part is followed by a *classification part*, which consists of a few density layers and ends generally with a softmax layer, in the same way as for a classic neural network.

### 2.1.1 Best practices for CNN design

#### Resolution and features maps trade-off

Generally, the first convolutional layers detect simple patterns of the input (simple geometric shapes in images) and the deepest convolutional layers detect high-level features (objects in an image, portions of the signal associated with a certain class in EEG etc.). At the same time, the downsampling layers are interleaved in order to reduce the resolution of the input. The resolution then becomes lower and lower for the deepest convolutional layers. Therefore, a common practice in CNN design is to increase the number of kernels for deeper convolutional layers. Because they can detect more complex patterns they should also be equipped with more filters [3].

#### Solving overfitting with regularization techniques

One of the major concern when training a CNN is the risk of over fitting the data. Due to the complexity of the models (in term of number of parameters for example) it often happens that the prediction accuracy on the training data becomes very good but decreases significantly on the testing data, this effect is even more prominent for a model trained on a small training set.

A solution that was implemented for the target CNN of this work was the use of Dropout regularization.

The Dropout technique consists of randomly ignoring a set of outputs from different layers, which equates of training a sub network with less parameters. At each iteration (batch) of the network, a different set of neurons is shut down. This allows to simulate the principle of *ensemble approximation*, that is, to average the predictions of several different networks, while actually do one training on one single network (with different combinations of closed neurons at each training iteration).

#### Mitigate the internal covariate shift and speed up the training

When the variance between two input features is highly different (in term of values they can take), the network will have difficulties to adapt its weights to the correct direction, especially if an important feature has a small range of values (0 to 10) and another less important feature has a large range of values (0 to 10000). This phenomenon is called the *internal covariate shift* and makes the cost function very asymmetric which in turn makes the process of backpropagating the gradients over the layers harder.

Therefore, each layer's output should be normalize to make the next layer learning process easier. Batch Normalization consists of normalizing (with mean 0 and variance 1) the input of a layer before each mini-batch training, i.e, every time the weights of the network change.

[3] by intuition we can imagine that in general, there are more different faces than they are different simple lines

#### Replacing pooling layers with convolutional layers

A last best practice that was experimented is the use of convolutional layers with a stride of 2 to replace max-pooling layers. This technique allows to perform a down-sampling of the input (i.e, dividing the dimension by 2) while leveraging the feature extraction ability of a convolutional layer.

## 2.2 Neural Architecture Search

The main idea of Machine Learning is to implement specific tasks - generally mastered by humans - in a machine, so that it can automates and outperforms the human ability for the task. Thus, since Turing's universal machine and the first neural networks, a whole bestiary of increasingly complex methods has come to enrich the field. The revolution that has taken place in recent years is that of Deep Learning which has made it possible to undergo breathtaking challenges, beating the best players of chess or of the game of Go, soon claiming to replace drivers, or even heal cancer. This breakthrough innovation obviously involves complexity, and it becomes more and more difficult for experts to design Deep Learning networks by hand. This is how emerged the need to automate the design of neural networks and more generally of all the methods belonging to Machine Learning. This research area is called Automated Machine Learning (AutoML [19]) and NAS is one of its component.

### 2.2.1 General definition

Neural Architecture Search is a sub-domain of AutoML that consists of automating the search for neural net architectures.

A good way to define a NAS algorithm is by specifying it in term of the three following dimensions:

- The *search space* defines the possible target architectures (called *children*) that the NAS algorithm can explore.
- The *search strategy* defines how the search space should be explored. The simplest case of search strategy would be a random sampling of the different children architectures.
- The *performance estimation strategy* is a crucial aspect in NAS, it determines which method the algorithm should use to estimate the accuracy of each generated child. Thus it has to be computationally efficient and reliable.

Figure 2.4 illustrates the interaction of theses three main aspects of a NAS. Following is a more indepth view of each component.

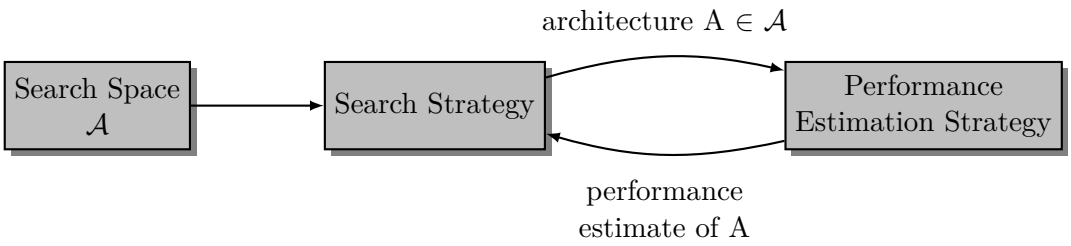



Figure 2.4: The NAS paradigm

### 2.2.2 Search space

The search space consists of all the neural network architectures, called *children* that the NAS can explore. In the case of this thesis, the CNN architectures that were searched are abstracted in terms of *cells* and *blocks* components, the network is a set of consecutive cells and each cell is made of one or several blocks. To see this more clearly, in figure 2.5 is an example of such a CNN. The network starts at the root with the input layer, then the input is fed to the rest of the network where each colored rectangle corresponds to an operation (a convolution operation or a max-pooling operation for example). Each set of same colour operations corresponds to a cell. And each cell is made of blocks that are numbered.

A block can be made of one or two operations, typically an operation is a layer of a CNN network (for instance, a convolutional layer or a batch normalization layer). When a block has only one operation, then it also has one input and one output. When it has two operations, each of them takes one input and their outputs are combined (generally with an element-wise addition layer). In figure 2.5, the only block with two operations is the block B1 of the first cell (green).

We consider two types of cells, the *convolutional* cells (or *normal* cells) and the *reduction* cells. Convolutional cells consist of convolutional layers that can be mixed with Dropout[18] or ReLu layers [21] to prevent overfitting, they use one or two operators per block. Reduction cells are used to reduce the dimensionality of the network, they generally consist of blocks of one operation, like *max-pooling* or *average-pooling* operation, for example, the red cell in figure 2.5 could be a reduction cell.

For this thesis, the search was limited to one pair of Normal/Reduction cells. This method is also used in [25] where they first train the pair on a small dataset (CIFAR-10) and, afterwards, by stacking several of theses pairs in a row, obtain a more complex architecture able to generalize on more complex datasets (ImageNet) by using the principle of transferability. Another important benefit is that it allows to search much faster than searching directly complex architectures.

In this thesis, we work with a single dataset, but stacking several pairs still allows to generalize better by bringing more complexity to the model, in this regard the final architectures constructed from the findings of the NAS algorithm follow this pattern where a maximum of three (identical) pairs are stacked on each other.

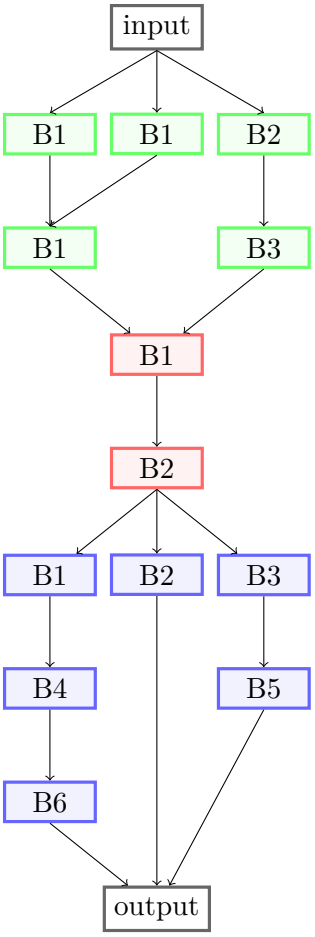



Figure 2.5: Example of a CNN made of cells and blocks

### 2.2.3 Search Strategy

The search strategy of a NAS defines how the algorithm will explore the search space.

The first search strategies in NAS date back to the 90s and mostly used Genetic Algorithms (GA) [10]. In GA, a population of architectures is randomly sampled and a looping algorithm inspired by natural evolution selects the best individuals, applies variations on them (mutation and recombination) and generates a new population of more fitted individuals. GA can be referred as an *extra-life* learning paradigm because, at each new generation, individuals die and some others arise. A more recent paradigm used as a search strategy is Reinforcement Learning which aims at improving the behaviour of an *agent* (*intra-life* learning) by taking into account a reward value based on its past and future actions. In recent years, the advent of Reinforcement Learning has shown promising results and seems to be used more consistently in research papers, this is why this paradigm was chosen here.

### Reinforcement Learning as a Search Strategy for NAS

In this section we explain what is Reinforcement Learning (RL) and how it was used in the thesis as a search strategy for the NAS algorithm.

#### Design

RL is a paradigm of Machine Learning, where an agent is trained in order to maximize its reward in a environment based on the actions it takes. The behaviour of the agent is guided by a *policy* which consists of all the decisions that the agent can take. This policy is based on a probability distribution which is updated in function of the future decisions and their corresponding reward. The search strategy for a NAS, can also be instantiated using this paradigm.

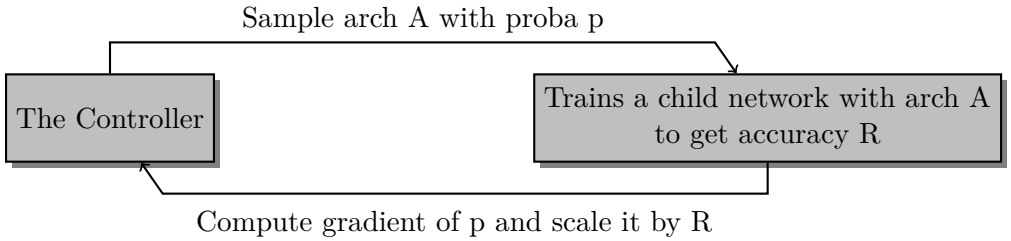



Figure 2.6: RL paradigm for NAS

In this case, the *environment* where the agent evolves is the *search space* as defined above. The agent is a recurrent neural network (RNN) called the *controller*, and the actions of the controller are the decisions that it makes to sample a child network architecture. Theses decisions correspond to the outputs of consecutive softmax layers. Figure 2.7 shows an example of the decisions outputed by such a controller. In reality, the controller does not directly outputs strings to express what decision it makes, but instead it outputs a probability distribution over a range of choices.

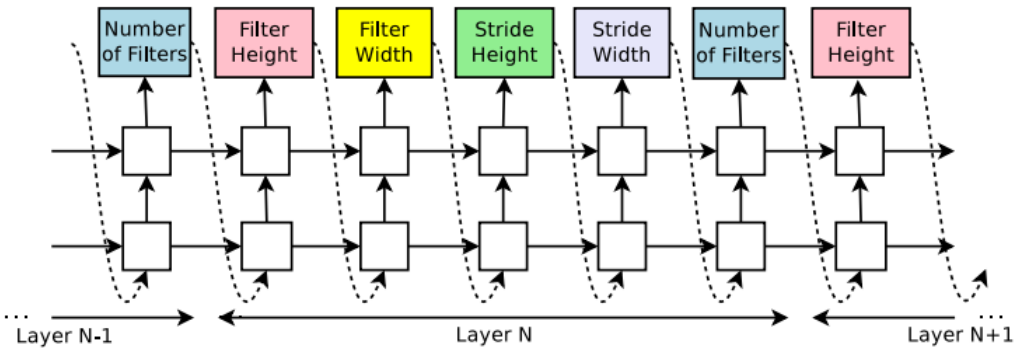



Figure 2.7: example of strings produced by a RNN controller

The update policy of the RL paradigm, corresponds to the updates of the controller

parameters (its weights). The one used in this thesis is called REINFORCE (Williams 1992 [22]), it computes the gradients of the RNN parameters so that they can be updated at each iteration. In a classic neural network, the optimizer is generally based on gradient descent, which computes the gradients in order to minimize a loss function, but in the case of REINFORCE, the opposite is happening, the loss function is replaced by a reward function, and the optimizer intends to maximize it. [4]

During the training phase of the controller, the generated children are trained and their accuracies are taken as a reward. The controller weights are then updated to maximize the expected value of the reward, expressed in equation 2.1.

$$J(\theta_c) = E_P(a_{1:T}; \theta_c)[R] \tag{2.1}$$

where, $\theta_c$ is the policy (weights) of the controller, $R$ is the reward (or accuracy of a child), and $a_{1:T}$ are the actions of the controller (all choices made to build the child network).

Directly computing the gradient of 2.1 is not feasible because the reward is a discrete value, thus, non differentiable. This imposes the use of a *policy gradient* method to iterately update $\theta_c$. Inspired by the Monte Carlo approach, the REINFORCE [24] method uses an estimation of the above quantity based on samples of the controller decisions, the update of the parameters goes as follow :

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_P(a_{1:T}; \theta_c)[\nabla_{\theta_c} logP(a_t | a_{'t-1):1}; \theta_c) R] \tag{2.2}$$

Which can be approximate to:

$$\frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} logP(a_t | a_{'t-1):1}; \theta_c) R_k \tag{2.3}$$

where $m$ is the number of sampled children at each iteration of the algorithm and $T$ is the number of architecture decisions made per child.

This update value can suffer from a high variance because of the consequent size of the search space. Generally an exponential moving average of the previous architecture accuracies is used as a baseline and subtracted to each reward:

$$\frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} logP(a_t | a_{'t-1):1}; \theta_c)(R_k - b) \tag{2.4}$$

This helps to center the rewards to lower values and partly solves the variance issue. We will see now, how a refinement of this formula allowed to reach even less variance and avoid an exploding gradient problem.

[4] in practice, we still use a minimizing optimizer, but we apply a negative sign on the reward function

### 2.2.4 Performance estimation Strategy

Performance estimation in the context of a NAS consists of estimating the performance of the children as accurately as possible. In a ideal world one should train each child long enough to retrieve its most realistic (representative) accuracy. For a NAS to be enough efficient and to learn fast, one must find ways to quickly estimate the accuracy of each child, one biggest challenge in this regard is to clearly separate between promising children and the others. For this, a new function that emphasize the quality of the returned rewards was designed.

#### New improvement of the reward function

The improvement on the reward function aimed at making the controller learn more efficiently, for this, a new function taking the accuracy as input was found.

The idea was to find a reward function that amplified the difference between low and high accuracies, in order to foster as much promising models and penalize 'no good' models during the learning of the controller. Instead of using the accuracy directly as the reward, one can use the output of an exponential function of the accuracy, thus amplifying the difference between low and high rewards. In figure 2.8 one can see the effect of using such a function.

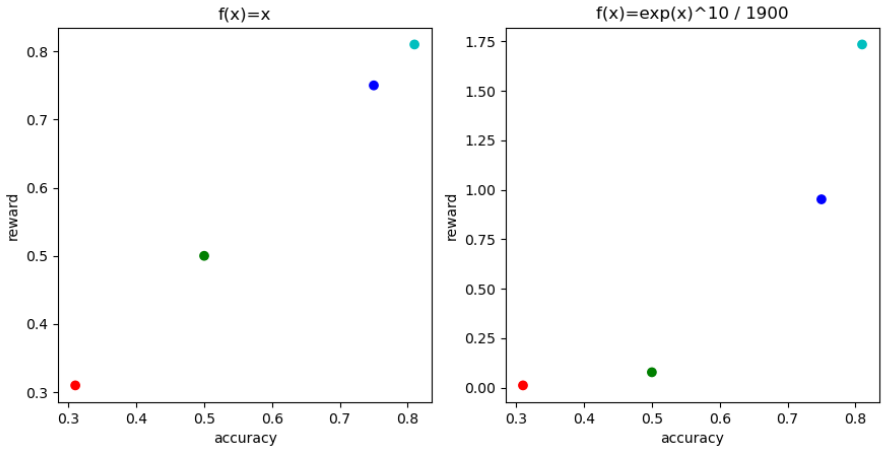



Figure 2.8: Comparing exponential reward function with identity function

This function is of the form:

$$Reward(x) = \frac{exp(x)^A}{B}$$

with A, B, x $> 0$ and where x is the accuracy and where A allows to amplify the difference between inputs, and where B allows to normalize the final value.

A more elegant function that allows for more control over its different characteristics was found:

$$Reward(x) = \frac{1}{(1 + exp(-x * A + B))} * C$$

If we can determine a threshold between bad and good accuracies (say 0.8), and if we set $\frac{A}{B} = threshold$ then the function changes its concavity at this threshold[5]. We can also increase the slope of the function around the *threshold* value by increasing the value of $\frac{A}{B}$, so by multiplying A and B with the same factor. Finally, we can increase the output of the function by a factor of $C$ which has the same effect of using a larger learning step for the backpropagation algorithm.

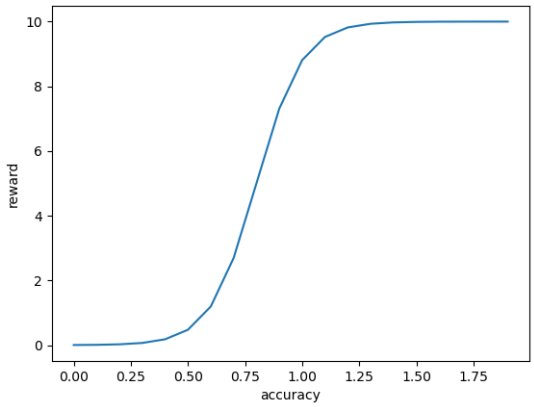



Figure 2.9: $Reward(x) = \frac{1}{(1+exp(-x*10+8))} * 10$

By bounding the values of our new reward function, we keep the variance of the ouput low, and by using an exponential-like shape, we are able to emphasize the difference between good and bad models. Both enhancements lead to a more efficient learning of the controller. A comparison between using the accuracy as a reward and using this function on top is given in chapter 5. An example of the reward function is given in figure 2.9.

[5] a function changes its concavity at point $x$ when $f''(x) = 0$ which is equivalent to set $\frac{A}{B} = x$

### 2.2.5 Implementation

The following pseudo-code shows the main loop of the NAS algorithm:

```
instanciate controller;
define number of epochs (nb_epochs) ;
define number of children sampled per epoch (nb_samplings) ;
for i in nb_epochs do
    expected_reward = 0;
    for j in nb_samplings do
        sample controller decisions (decisions) ;
        for dec in decisions do
            expected_reward -= log(Proba(dec));
        end
        create corresponding child architecture;
        train child;
        compute child accuracy (child_acc);
        expected_reward *= (child_acc - baseline);
    end
    expected_reward /= nb_samplings;
    update controller weights with ∇(expected_reward) ;
end
```

**Algorithm 1:** main loop of the NAS

The code was structured in a Object Oriented way, and split into three main classes:

- The *controller* class corresponds to the controller RNN used as the agent in the Reinforcement Learning paradigm. It takes as class arguments: the number of convolutionnal and reduction cells, the number of available operations for each type of cells and the number of pairs of normal/reduction cells that should be stacked. The main function of this class is the *generateController* function which actually constructs and returns the RNN network. It consists of a loop over the choices to be made (number of cells etc.), and for each choice it stacks the following sequence of layers: a Lstm layer, a Dense/Softmax layer, and a Lambda layer (which randomly samples the class to be chosen, based on the probability of the Softmax).

- The *Cell* class takes as arguments the type of cell to generate (normal or reduction) and an array of the blocks that form the cell. The *generateCell* function, loops over the blocks array and for each of them instanciates the Block class, and stacks the blocks with each others.

- The *Block* class, takes as argument the inputs and the operations of the block, and constructs the block by combining the inputs (which are Keras[4] layers object) with the operations (also Keras layers objects) and finally by combining both operations (with the Keras layer *Add*).

The code was implemented with Keras [4] as it is a well known library for deep learning, practical in use and well documented. It contains the Functionnal API, which is a flexible and easy-to-use way to build models, where each layer can be instanciated and stacked on each others in one line of code.

Here is a set of very useful Keras and TensorFlow [1] functionalities that were used in the code:

- the *GradientTape* class was used to record all operations during the sampling and training of child network, in order to compute the gradient to update the controller weights.
- the *MirroredStrategy* class was used to enable multi-GPU processing for the training of the children networks.
- *load_weights* and *save_weight* functions from the Functionnal API were widely used, especially for the ENAS extension of the algorithm (see below).
- the *Dataset* class was used to build an efficient input pipeline to feed the child network with.

**Extension of the NAS algorithm: Efficient Neural Architecture Search**

At each iteration of the algorithm a child must be sampled and trained, this consitutes the main bottleneck of a NAS algorithm in term of computation efficiency. To solve this issue, different improvements have been made in recent years for RL-based NAS algorithms. For this thesis, the choice was oriented toward a version called the Efficient NAS (ENAS) algorithm. [15].

**Design**

In the ENAS paradigm, the search space is represented as a Direct Acyclic Graph (DAG), where each node is an operation (a layer) and each edge is the flow of data between layers. Each child sampled by the controller is a sub-graph of the DAG. This abstraction helps to understand that for a given search space, it happens often that some children have in common the same edges or/and nodes of the DAG, in other words, that they share the same architectural elements. Therefore, ENAS improves the training phase of the children networks by forcing them to share the weights for the parts they have in common. This technique uses the principles of transfer learning, leveraging the beneficial impact on the training phase by sharing weights between similar (but different) architectures [17].

In figure 2.10 we can see an example of the DAG representation for a search space consisting of six operations and where the red edges and the green nodes that they link represent one of the children architectures. In this child, the root of the network is node 1 (so, it is the $1^{st}$ layer of the network) which is then fed as an input to node 2 and so on.

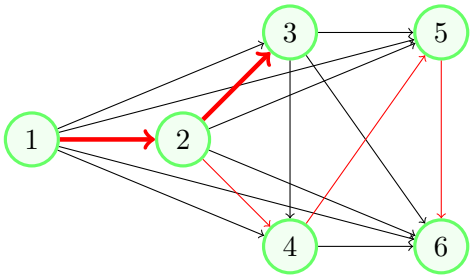



Figure 2.10: ENAS DAG with a child sub-graph

In ENAS, when this child network is trained during an iteration of the algorithm, the weights corresponding to these red edges are saved, and reused later for the training of another child which shares theses edges. For instance in figure 2.11 the child will reuse the weights of edges 1-2 and 2-3 that were already trained for the child on figure 2.10.

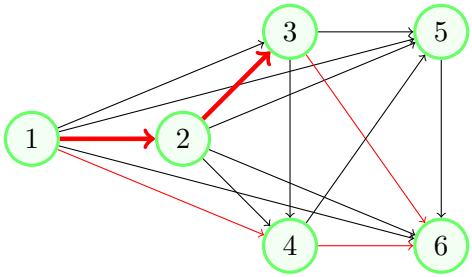



Figure 2.11: ENAS DAG with another child sub-graph

### Implementation

The only difference with the classic NAS implementation is on the weights sharing part, which is new here, but luckily, Keras makes it easy for us. To illustrate this,

here is again the main loop expanded with the weight sharing code:

```
File containing all weights (total_weights) ;
for i in nb_epochs do
    expected_reward = 0;
    for j in nb_samplings do
        create corresponding child architecture;
        load into child the weights from total_weights ;
        train child;
        update total_weights with new weights from the training ;
    end
    expected_reward /= nb_samplings;
    update controller weights with expected_reward ;
end
```

**Algorithm 2:** Implementation of the ENAS algorithm

**New improvement on the weights sharing**

Each time a same child architecture is trained during the NAS training, a new, different accuracy is returned. But the reference accuracy should be the highest one found among all trainings, because it is the best performance so far that the child is able to reach. If we only use as reward, this highest accuracy instead of the accuracy of the current training, the value of this reward will show much less variance and be closer to the true value of the child, this way, we improve the learning process of the NAS.

In chapter 5, we show with an experiment, the benefits of the two previous improvements.

## 2.3 Related work

The preparation phase for this thesis consisted of investigating different related works on EEG classification with CNN. The most important one, was the work of Ansari et al. who created a state-of-the-art CNN for the Quiet Sleep classification problem, the results of this work constituted a reference and a starting point for the thesis. In a second time, it was also important to look for other research paper related to EEG classification with CNN in order to get a broader understanding of the search space, and maybe to find original ideas.

### 2.3.1 CNN from Ansari et al.

Ansari et al. developed a convolutional network (figure 2.12) to perform 2- and 4-class sleep-stage EEG classification, for the present thesis the focus was only on the

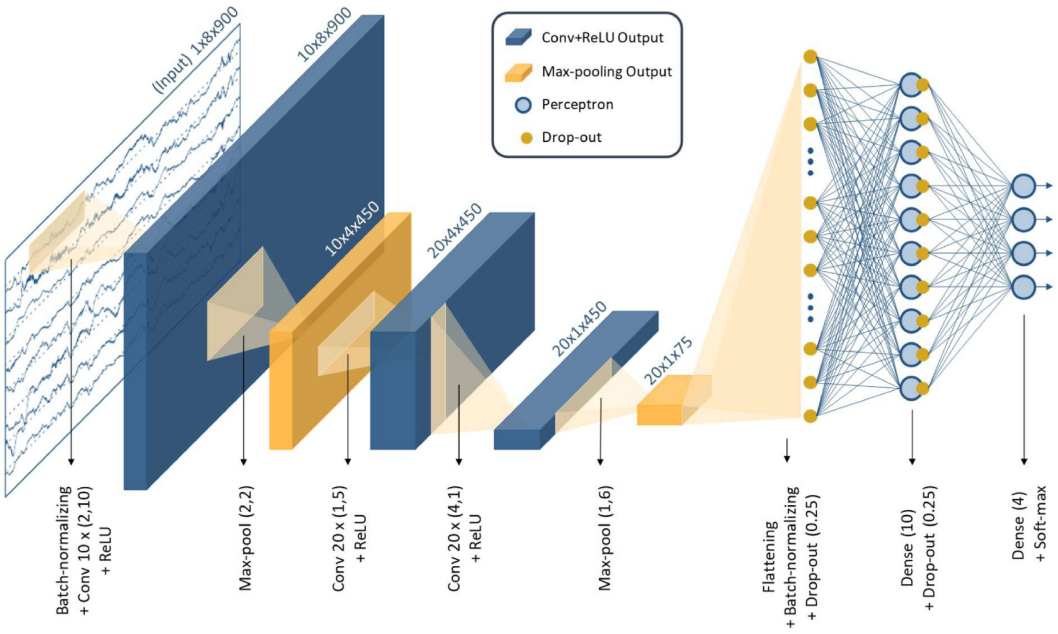



Figure 2.12: CNN network from Ansari et al

2-class problem, i.e, detecting the Quiet Sleep stage. The CNN proposed by Ansari et al. was designed with the following good practices:

- using multiple convolutional layers to extend the complexity of the network, and therefore to increase the ability to capture potential features
- using pooling layers to reduce the number of trainable parameters (and to controle overfitting)
- using bach-normalization layers to standardize the extracted feature maps
- using drop-out layers, to increase the generalization of the dense layers

The input of the network consists of an array in two dimensions, representing the EEG signal of 8 electrodes (spatial dimension) recorded over 30 seconds (or 900 timesteps).

The $1^{st}$ layer is a convolutional layer and it aims at capturing the best the two dimensions of the input. On the spatial dimension, it scans the channels 2-by-2 [6] and in the temporal axis, it scans the time steps 10-by-10 (a third of a second) as it represents a good time frame for capturing local features without being too short to capture only EEG noise.

The other layers alternate with down-sampling layers (max-pooling) to reduce the dimension of the inputs, and convolutional layers to capture higher features.

[6] Ansari infers that this allows to emphasize the contralateral neural activities, but with this setting and according to the 10-20 system, the pairs of electrodes in this case are direct neighbours

The last part of the network is the classification part, it first flattens the output of the last max-pooling layer and uses Dense layers for the classification task, the output of the network is a Softmax layer consisting of two or four units for the 2-class and 4-class classification problems respectively.

The network, when released, was state-of-the-art on the Quiet Sleep classification and demonstrated a Kappa score of 0.76 (see Chapter 5 about metrics).

### 2.3.2 Other works on EEG classification with CNN

The low cost and non-invasive setting provided by EEG opens the way to a wide variety of experiments on brain related research. Thus, many research papers deal with EEG classification with CNN, and some of the ideas are worth mentioning here as they helped thinking about the problem.

Some architectures combine EEG with other inputs, such as wavelet transform in order to convert the input to time-frequency domain [13], indeed spectral analysis is known as being the most powerfull technique for EEG analysis [20].

Brain Computer Interface (BCI) research also shows a lot of interest in EEG classification, a model called *EEG-Inception* network [23] has reached an accuracy of 88.6% on Motor Imagery binary classification, by using data augmentation to reduce overfitting.

Other papers related to BCI, show state-of-the-art results with raw EEG signals recorded over several electrodes [12], which is the case for the present thesis. Thus, a useful information that was exploited for the thesis was to separate the convolution filters on the temporal and spatial axis (channel axis) of the EEG. One of the main technique to deal with EEG noise it to use spatial filtering, i.e, to look for interesting combinations of electrodes that would yield to the best accuracy of the signal. Various methods already exist to find the best combinations such as: bipolar filtering, common average reference (CAR), Laplace filtering or common spatial patterns [3]. Thus, empowering feature detection on the spatial axis of EEG can also be used to find good channel combinations, this technique will be discussed in Chapter 5.

Regarding the actual types of layers used for EEG classification in this thesis, other sources from the literature ([12] [13] [6]) indicated similar choices as previously mentioned. In [12], the main assumption to yield to state-of-the-art results is again to separate temporal and spatial filters, their architecture focuses on the detection of time-related features and consists of 5-layer CNN among which four are convoluted along the temporal axis and one along the spatial axis. In order to cope with the vanishing gradient problem, they use Leaky ReLU for the activation function as well as dropout and batch normalization to reduce the risk of overfitting, they also choose a learning rate of 10e-5 to reach more optimal solutionss. Finally, they use Adam as an optimizer for the gradient descent algorithm, and they based it on an adaptive learning rate which speeds up the convergence of the model.

# Chapter 3

# Data

This chapter introduces the data used for the experiments. A first part discusses the type of the data, which are EEG recordings, then, a second and third parts explain how the dataset was made and formatted.

## 3.1 Electroencephalograms

An electroencephalogram (EEG) is the measure of the electrical activity or voltage, detected by an electrode placed on the scalp of an individual.

The brain activity generates an electrical flow constituted of ions moving in and out of the neurons and along the axons. Pyramidal neurons, which are perpendicular to the cortical surface are responsible for the cortex electrical activity by sending negative or positive charges toward it.

This electrical activity penetrates through the skin and finally reaches the electrodes used to capture the signal. But the impedance of the skin tissue makes the amplitude of the signal and its spatial resolution lower, this problem is referred as *volume conduction*. The loss of information induced by volume conduction can be a substantial obstacle for specific applications such as brain computer interfaces (BCI), but the use of EEG is sufficient for a lot other applications, such as, detecting sleep-stage state in newborn babies.

Other non-invasive techniques have been considered to monitor brain activity in babies, like Near-Infrared Screctoscopy with continuous wave (CW-NIRS) [7], where the reflexion of light pointing toward the skull provides information about the rate of oxygen in blood (oxygenated hemoglobin) in a specific location of the brain. Although not expensive, this solution suffers from different drawbacks that still rank it behind EEG in term of quality of the signal, for example, a low spatial resolution (worse than EEG), an underlying model (Beer-Lambert) which is not enough complex and a poor temporal resolution with a sampling frequency of maximum 6Hz.

EEG on the other hand, offer a very high temporal resolution (in the order of milliseconds) and a much longer history of practical use.

## 3.2 Composition of the dataset

The dataset provided for this thesis was first used by Ansari et al. [2] for their work on EEG classification with CNN which this thesis is based on. The dataset was issued by the NICU of Leuven University Hospital, in Belgium. It was obtained on parental consent and validated for research by the ethics committee of the hospital.

Each recording consists of nine EEG signals corresponding to nine electrodes placed on the infant skull: Fp1, Fp2, C3, C4, Cz, T3, T4, O1, and O2 with Cz used as reference. The notation of the electrode is based on the 10-20 system [9], and an illustration is shown in figure 3.1[1].

The EEG were recorded with the BrainTR system (OSG BVBA Rumst, Belgium), at 250Hz, on infants that did not present any abnormal neurodevelopmental symptoms at the age of 9 months and 24 months.

From theses recordings two datasets were created, but only one is used here, the *preterm* dataset. It consists of a total of 97 EEG recordings that were retrieved from 26 preterms newborns (i.e born before 32 weeks of PMA) between 2012 and 2014. At least two recordings were performed on each of the babies.

The dataset was prepared for 2-class classification and the labelling of the QS segments was carried out by two EEG experts (KJ and AD) upon consensus, and the non-QS segments include all the other phases of wakefulness and sleep (AS) as well as indeterminate segments (IS).

## 3.3 Formatting of the data

The *preterm* dataset was split into a training and a testing subset. The training subset consists of 32272 segments and the testing subset of 26736 segments.

The data was bandpass filtered (between 1Hz and 15Hz) and down sampled at 30Hz. Each segment was labeled and has a duration of 30 seconds without overlap. Figure 3.2 shows an example of such a segment, only two channels are presented here, but there are 8 channels in the actual data.

[1] https://www.researchgate.net/figure/Nomenclature-des-electrodes-dEEG-de-lInternational-10-20-System_fig2_316647177

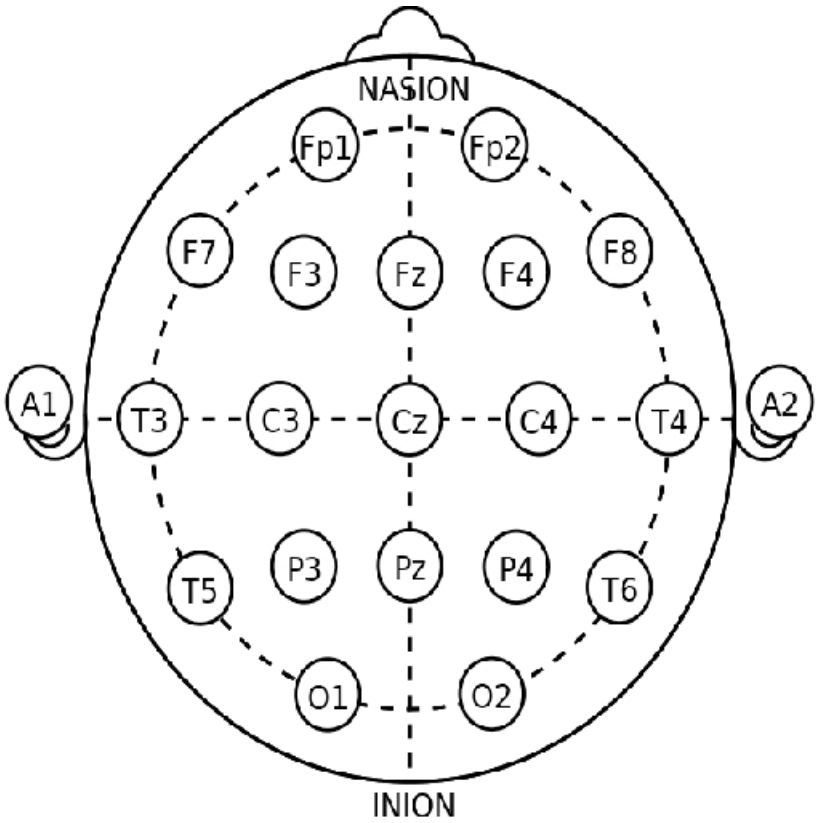



Figure 3.1: 10-20 system

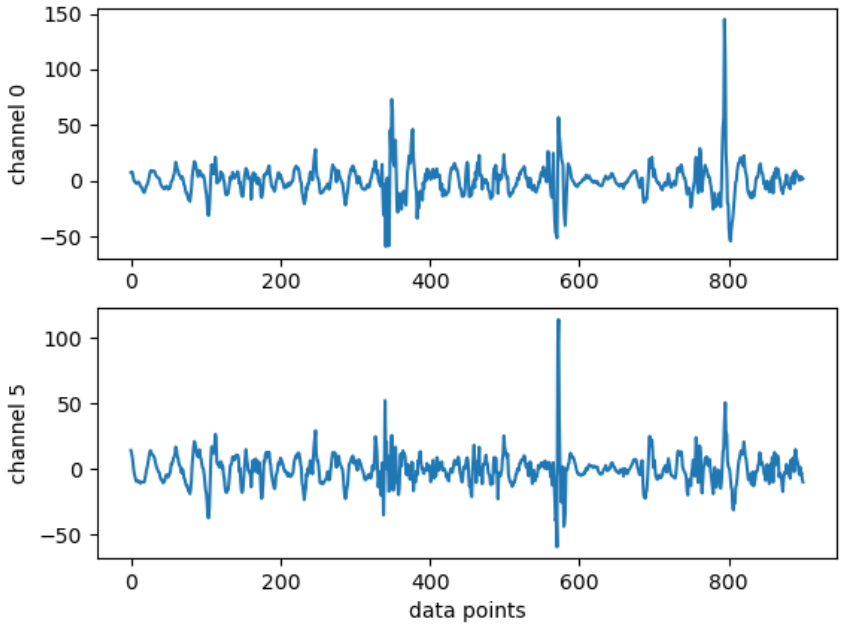



Figure 3.2: 2 channels recording over 30 ms (900 datapoints)

# Chapter 4

# Methodology

In this chapter, a first part discusses the methodology developed during the thesis for using the NAS algorithm (i.e how to train the controller) and optimizing its performances with preliminary steps. A second part shows a concrete example and proves the validity of the methodology. A third part discusses the process for assessing the quality of the final architectures returned by the NAS controller after training.

## 4.1 Proposed Method

Taking into consideration all previous remarks, the search of new networks with the NAS algorithm was conducted with a structured methodology which consisted of the following steps:

1. Choosing the search space, in term of choice of the layers and structure of the children networks
2. Preparing the search by adjusting critical hyper-parameters
3. Conducting the search
4. Re-adjusting the parameters from 2) and iterating

### 4.1.1 Choosing the search space

The search space can be abstracted in two sub spaces, the search space over the layers constituting the blocks and cells of the children and the search space over the architectures (how the layers are combined together to form the blocks and cells).

#### Search space over the layers

At each iteration the algorithm creates a child architecture which consists of two fixed part, the input and the output layers, and of a dynamic part sampled from the controller, the middle layers. Both parts can be adapted to deal with different types

of networks, although the thesis focused on CNN architectures.

- **Fixed layers**: for each child , the same starting and ending layers are kept during the search, only the middle of the network are explored through the search space of layers.
- **Dynamic layers**: the middle part of each child are explored by the NAS in order to find the best layers and the best combinations between them. To build a valuable search space of the layers, different sources can be explored, preferably one should choose layers that are similar or identical to already successful layers found in the literature.

**Search space over the structure**

As explained in Chapter 2, a search for the best Normal cell followed by a Reduction cell was performed. Each type of cell is a combination of blocks. For the Reduction cell the blocks are simply stacked consecutively, without skip connections and with only one operator per block. For the Normal cells, we defined four sub categories of the search space (*schemes*), to allow for more freedom regarding the combinations between blocks.

- Scheme 1: cell without skip-connection and with only one operation per block
- Scheme 2: cell with skip-connections and with at most one operation per block
- Scheme 3: cell without skip-connections and with at most two operations per block
- Scheme 4: cell with skip-connections and with at most two operations per block

An example of each scheme as well as an example of a Reduction cell and an example of the entire network is given in the Appendix B.

### 4.1.2 Preparation of the Search

A NAS algorithm needs hours and often days to show tangible results, so, once the search space has been decided we need to adjust some of the parameters of the NAS to avoid useless long lasting computations leading to poor results. The parameters to set are of two categories. The first one is the maximum number of epochs allowed per child training, a minimal value of this number of epochs allows to increase the speed of the algorithm, i.e, to minimize the time between two iterations. The second one is the choice of the reward function to optimize the convergence of the controller, so that it samples increasingly better networks in the most efficient way.

To find the values for theses parameters, we implemented several testing tools that enable to capture an approximation of certain characteristics of the search.

#### Optimizing the speed

At each iteration of the NAS, one child architecture is sampled and trained, this training process is the bottleneck of the NAS is term of speed iteration. To reduce the training time of the children, we have to determine a minimal number of epochs sufficiently large to allow for a significant distinction between promising networks and bad networks.

The process of finding this best number of epochs, is in three steps:

1. run a function (called *best_epochs* that samples a number of children (the more the better), trains them over a large enough number of epochs, and plots the evaluation losses and accuracies.

2. from 1) we can assess what minimum amount of epochs are needed to see a distinction (the clearest the better) of the children performances in term of evaluation loss and evaluation accuracy.

3. in order to be sure about the choice of the number of epochs, we can assess the validity of the retrieved accuracy (validation accuracy) by plotting for each trained network, the metrics values (accuracy and kappa score here) calculated on the test set.

#### Optimizing the convergence

As explain in 2.2.3, in order to make the controller learn as efficiently as possible one should adapt the reward function in such a way that it would give very high reward values for the best models and very low reward values for the other models.

To do so, one can sample many children, compute their accuracies, and take the highest one as a threshold, then, one should choose an exponential reward function that outputs large values above this threshold and very low values below.

#### Testing the parameters values by emulating the NAS

To validate the choices of the two parameters (the minimal number of epochs and the appropriate reward function), a function that emulates the training of the controller was created, it does not train the children themselves, but first draws a distribution of the validation accuracies returned by the children (over 1000 iterations for example). Then, by training the controller and affecting at each iteration an accuracy sampled from this distribution, one can assess grossly if the controller will eventually sample better architectures and with which efficiency.

### 4.1.3 Conducting the search, readjusting and re-iterating

The testing function for evaluating the NAS performance gives a gross estimation, and the NAS results are likely to be quite different, although they follow the general trend of the estimation.

Therefore (and this is shown in the next part), it is often necessary to run the algorithm not completely and to compare between the real results and the expected ones, this way we can re-adjust the parameters (especially the reward function) and run the algorithm again.

## 4.2 Experiments

In this section are first shown the choices that were made in term of the search space, and then an example of how to apply the method.

### 4.2.1 Choices of the search space

#### Fixed layers

In figures 4.1 and 4.2 are the starting and ending parts of the architectures of the children that were searched by the NAS, they correspond to the beginning and ending layers of Ansari et al. network. The "input" part reshapes and normalizes the input data, and the "output" part performs the classification.

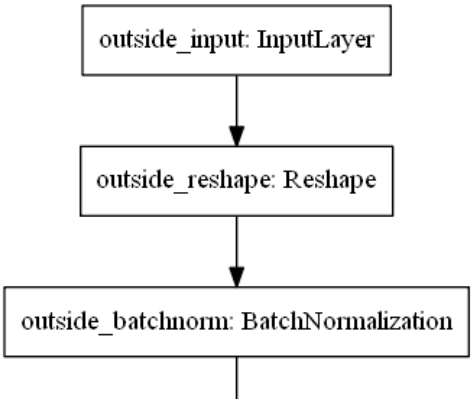



Figure 4.1: starting of each child

#### Middle layers

Layers from the CNN of Ansari et al. and from the literature (all introduced in Chapter 2) were used as a base for the search space.

**Layers from Ansari et al. Network**

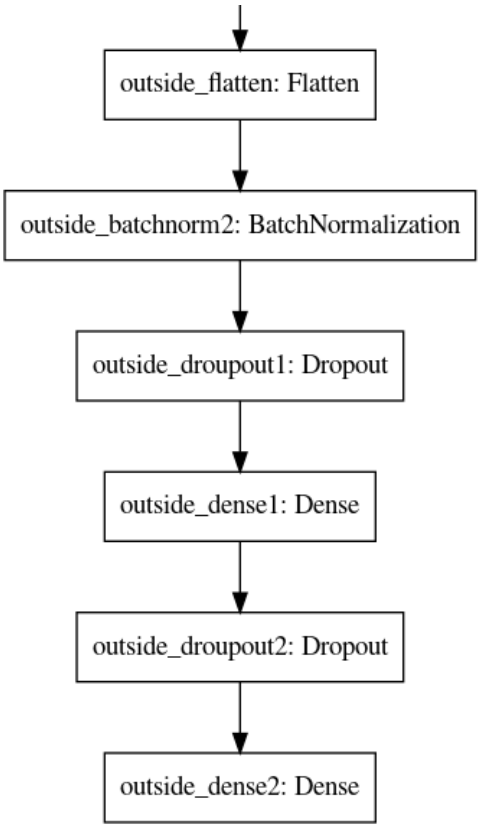



Figure 4.2: ending of each child

A hyperparameter search on the Ansari network was performed in order to determine the best combinations of parameters, in particulaer the first layer (convolutional layer) was investigated separately, to test for best values of parameters regarding the channel and time axis. The results of this search as well as an update of the Ansari network are given in chapter 5.

#### New layers found in literature

The ideas elaborated in Chapter 2 were used in order to expand the search space. A key idea when working with EEG signals was to exploit the two dimensions of the input, i.e, the spatial axis which consists of the 8 electrodes channels and the temporal axis which consists for each training example of 900 datapoints (30 seconds recordings).

The base network of Ansari et al. used a convolutional layer to filter for patterns involving two consecutive channels on the spatial axis which did not cover all possible combinations between two channels, therefore this search of other combinations was also performed with an hyperparameter search on the $1^{st}$ layer of the Ansari network.

The spatial axis was explored in the combinations between channels but also in the number of channels.

Another idea was to test for convolutional layers with a stride of 2 to replace max-pooling layers. Indeed, this technique allows to perform a down-sampling of the input (dividing the dimension by 2) while leveraging the feature extraction ability of a convolutional layer. A last idea was to use average-pooling layers in place of max-pooling layers, indeed, the former should be more suited in case of EEG where averaging among the different channels can help removing the noise.

#### Structure of the network

All four schemes were explored during the experiments, one can see how they look like in Appendix B.

### 4.2.2 Example of the preparation step

#### Optimizing the iteration speed by choosing the right number of epochs

To find the minimal number of epochs required to retrieve a representative accuracy of each child one can sample a number of children and plot the evaluation accuracy and loss over a large enough number of epochs.

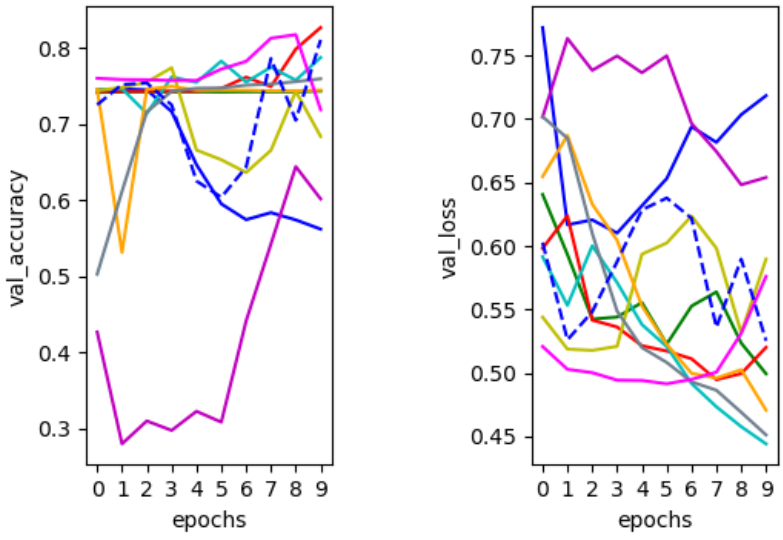



Figure 4.3: Accuracy and Loss vs Nber of epochs

Figure 4.3 shows the results. The idea is to choose a number of epochs from which validation accuracies are grossly differenciated. If we choose 4 epochs for example, we see that the relative position of accuracy of each child among the others represents more or less its position if we consider the max accuracies. For instance, the pink line at 4 epochs is above the other lines which is also the case at the 8th epoch, whereas the blue line is down and stays down until the 10th epoch.

If we choose a number of epochs large enough, during the NAS training, as we keep and share the best weights among children, after a while, the differenciation between accuracies should appear much clearer.

### Validating the choice of number of epochs

One can validate our choice for the minimal number of epochs by assessing the correlation between the validation accuracy and the accuracy on the test set.

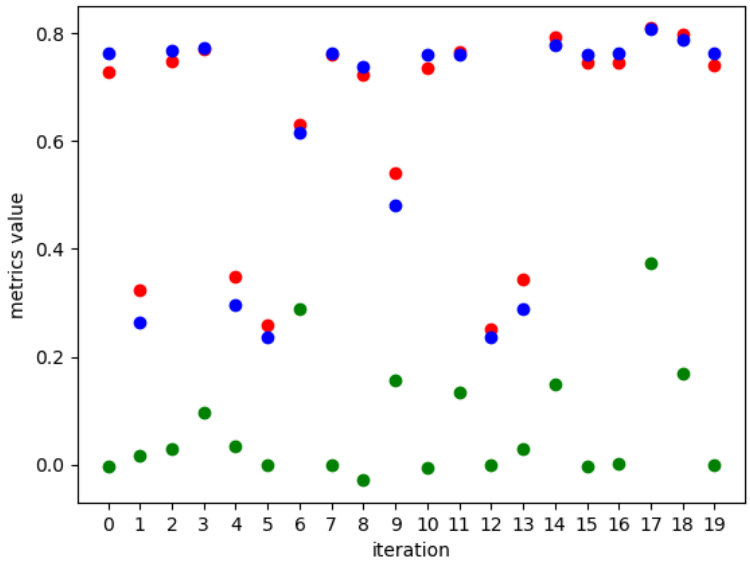



Figure 4.4: train vs test metrics

Figure 4.4 shows in red dots, the validation accuracy (on the train set), in blue dots the accuracy on the test set and in green, the kappa score on the test set. We clearly see that the accuracies on both test set and train set are correlated (dots are very close to each others). The kappa score on the other hand is a more demanding metrics, because it is also strongly influenced by the divergence between the real and estimated values, so slight differences will have a bigger (negative) impact on its value comparing to the value of the accuracy.

At least, one can notice that the highest values of the kappa score correspond to the highest values of the accuracy on the training set. As our NAS algorithm was

optimized to foster networks with highest validation accuracies it follows that these networks should show the best kappa scores as well.

### Optimizing the convergence with an adapted reward function

Figure 4.5 shows a histogram of the distribution of accuracies computed over 1000 children, each bar corresponds to a group of accuracies centered around a mean, for example the left bar shows that 10% of the children with the lowest accuracies have a mean of 0.31, whereas the right bar show that 2% have an accuracy centered around a 0.81 mean.

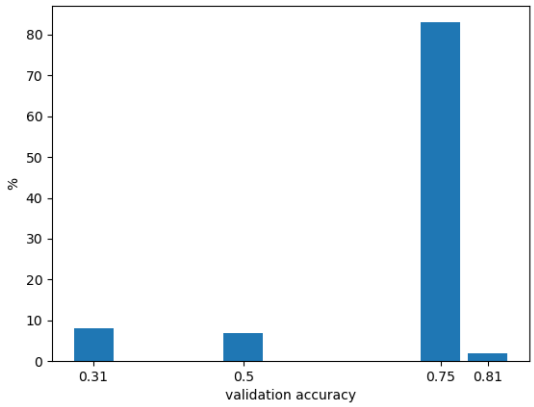



Figure 4.5: distribution of accuracies

We want to find a reward function that fosters accuracies in the 0.81 area (and above) and discriminates accuracies below.

In figure 4.6 [1], we can see the output of the identity function on the left and the output of the adapted reward function on the right, taking as input the mean accuracies from figure 4.5.

Finally, one can use the function for estimating the performances of the NAS with and without the adapted reward function, the results are shown in figure 4.7.

Figure 4.7a does not show any progress of the NAS, although it can looks worrying considering its actual performances, the reason is that the distribution of accuracies for this specific example shows that most than 90% of the architectures sampled have an accuracy centered around 0.75 which indeed cannot help the NAS to find better architectures, hence the importance of using an adapted reward function.

[1] the reward function in this figure is 1st version that was designed, but it still shows what is intended, i.e, a clear separation between good and bad accuracies

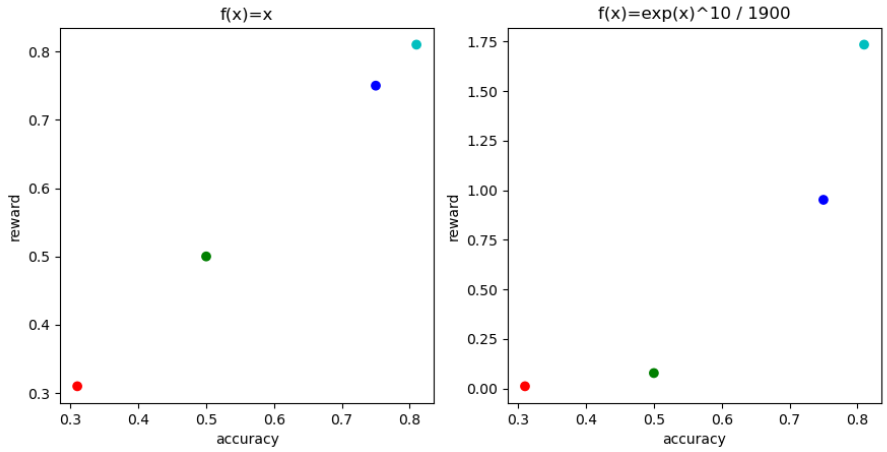
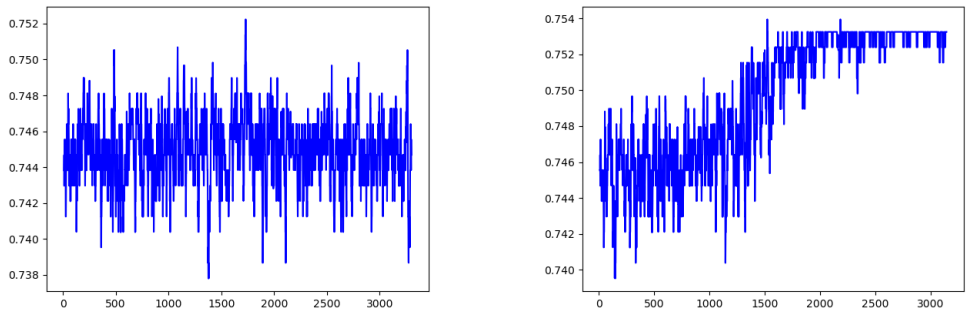



Figure 4.6: Comparing reward function with identity function




(a) Identity function reward

(b) Adapted function reward

Figure 4.7: Accuracies VS Time

## 4.3 Assessing the final results of the NAS

Once the training of the controller shows a plateau of the accuracies returned by the children it samples, we can stop the training and use the controller to only sample architectures.

The architectures in our case consist of a CNN (see appendix B), as discussed in section 2.2.4 the NAS algorithm searched over a pair of cells, a Normal cell followed by Reduction cell. In order to obtain a more complex architecture able to generalize more, we can iteratively stack this pair and for each stack, train the total network. In the experiments of the thesis, the final networks trained consisted respectively of 1, 2 and 3 identical pairs stacked on each others, where for each new pair the number of filters in the convolutional layers was doubled. An example of such a network is also given in appendix B. The training of each stack was performed with

a k-fold cross validation where k started at 10 and was doubled at each new stack, as more parameters were to be trained than for the previous stack. In short, this process consist of a double loop, the outer loop being over the number of stacked pairs and the nested loop being over k. Finally, after each iteration of the k-fold cross validation, the kappa score on the test set was computed and the best kappa score so far was saved.

Another type of assessment that was performed is to evaluate statistically what kind of layers or combinations between layers occurs most often in the sampled children. For this, a function was developed that samples N number of children and returns two dictionaries, one with the layers types as keys and the other with the blocks types as keys. For both dictionaries, the values represent the occurrence of the corresponding key.

# Chapter 5

# Results

In this chapter, an introductory part gives the definitions of the metrics used to assess the performance of the NAS. In a second part a hyperparameter search on the Ansari network is presented, showing combinations of parameters leading to improved results and to a choice of layers for the NAS search space. In a third part the algorithm developed for the thesis is benchmarked against a classic NAS algorithm, showing a clear increase in performances of the former. In a last part, the final results of the NAS algorithm are shown, first in term of mean accuracies and kappa scores of the children sampled from the different searches spaces (schemes) and finally in term of the interesting layers and combinations of layers that appear in the theses children.

## 5.1 Metrics

The metrics used to benchmark the findings of the NAS are the Cohen-Kappa score, and the prediction accuracy.

### 5.1.1 Prediction accuracy

The prediction accuracy is the percentage of correct predicted examples by the considered network, in the case of this work it corresponds to the accuracy returned by the CNN for predicting if an EEG segment matches with a QS segment or not.

### 5.1.2 Cohen-Kappa score

The Kappa score is a statistical measure of the agreement between observers during a classification task. The formula is the following:

$$\mathcal{K} = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}$$

Where $Pr(a)$ is the agreement between observers and $Pr(e)$ is the probability of a random agreement. If $\mathcal{K}$ is equal to 1, it means that the observers agree for all choices. Here, the two "observers" are the predictions made by the network and the real labels of the EEG segment. All the Kappa scores below were obtained after a post-processing of the data consisting of applying a moving average of size 6 on the vector of prediction labels. This is a process that was also applied in Ansari et al. paper as it enables to remove "outsider" predictions, i.e predictions that are surrounded with opposite ones. Indeed, sleep stage transitions are smooth and it cannot happen for a child in a non-QS phase, to go in a QS phase for 30 seconds and to eventually go back in the non-QS phase.

## 5.2 Work on the Ansari Network

### 5.2.1 Hyperparameter search

A search was conducted on all possible hyperparameters on the CNN from Ansari et al. (2.12), without modifying the structure of the network nor the dimensions of each layer. The rows on table 5.1 are ordered from the first tuned layer of the network (top row) to the last one (Dropout 2 row). The accuracy here refers to the evaluation accuracy computed during the search. Only the 4 best configurations are shown here.

| Layer | hyper-param | Network 1 | Network 2 | Network 3 | Network 4 |
|---|---|---|---|---|---|
| Conv 1 | #filters | 10 | 30 | 20 | 20 |
| | dilation rate | 4 | 4 | 1 | 1 |
| Conv 2 | #filters | 40 | 40 | 40 | 30 |
| Conv 3 | #filters | 10 | 40 | 10 | 10 |
| Dropout 1 | rate | 0.35 | 0.25 | 0.25 | 0.35 |
| Dense 1 | #units | 10 | 15 | 15 | 10 |
| Dropout 2 | rate | 0.35 | 0.39 | 0.45 | 0.40 |
| Learning rate | | 0.001 | 0.001 | 0.001 | 0.001 |
| Evaluation accuracy | | 0.90 | 0.89 | 0.89 | 0.88 |

Table 5.1: Hyperparameter search results on Ansari et al. CNN

After this search, the three best networks were retrained with a K-fold cross-validation (with K=600). Table 5.2 shows the mean accuracy that was computed after each cross-validation and the Kappa score obtained (both evaluated on the test set).

We see that Network 2 performed best in our experiments.

| | Network 1 | Network 2 | Network 3 |
|---|---|---|---|
| Mean accuracy | 0.82 | 0.91 | 0.88 |
| Kappa | 0.72 | 0.78 | 0.76 |

Table 5.2: LOSO cross validation accuracy and Kappa score on the 3 best networks from the hyperparameter search

### 5.2.2 Creation of the search space from the Ansari network

Another hyperparameter search was conducted specifically on the first layer (Conv(2x10) layer) to find the best combinations of channels and of time steps. Table 5.3 shows the evaluation accuracies obtained with different combinations of dilation rate and number of channels. Table 5.4 shows the evaluation accuracies obtained with different time range.

| dilation rate \ # channels | 2 | 3 | 4 |
|---|---|---|---|
| 1 | 0.89 | 0.79 | 0.78 |
| 2 | 0.86 | 0.81 | 0.78 |
| 3 | 0.85 | 0.80 | 0.80 |
| 4 | 0.91 | 0.77 | 0.79 |

Table 5.3: evaluation accuracy on the channel axis configurations ($1^{st}$ layer)

| time range (datapoints) | 2 | 5 | 10 | 15 |
|---|---|---|---|---|
| eval. accuracy | 0.68 | 0.89 | 0.85 | 0.83 |

Table 5.4: evaluation accuracy on the time axis configurations ($1^{st}$ layer)

We see that on the spatial axis, the best combination of values is to take two channels with a dilation rate of 4, which means two channels that are separated by 3 other channels. And on the time axis, the best solution is a time step value of 5 datapoints (about 0.17 seconds).

In the end, the search space over the layers was a mixte between layers found by the hyperparameter search on the Ansari network, layers suggested by the literature and layers following the good practices introduced in 2.1.1. We annotate the first with *Ansari*, the second with *literature* and the third with *good practice*.

The following layers are for the Normal cell:

- Conv((2,10)), a convolutional layer with filters of dimension 2 x 10 (*Ansari*)
- Conv((2,10), dilation_rate=4 ) (*Ansari*)
- Conv((2, 5)) (*Ansari*)
- Conv((2, 5), dilation_rate=4) (*Ansari*)
- Conv((8,1)) (*literature*)
- Conv((1,900)) (*literature*)
- Dropout(0.2) (*good practice*)

And these layers are for the Reduction cell:

- MaxPool((2,2)), a max pooling layer with a 2 by 2 spanning window (*Ansari*)
- MaxPool((1,6)) (*Ansari*)
- Conv2((2,2), strides=2), a convolutional layer with a stride of 2 (corresponds to a shift of 2 units on both axis, which leads to an output dimension divided by two) (*good practice*)
- Conv2((1,6), strides=2) (*Ansari + good practice*)

## 5.3 NAS efficiency

In this section are compared at first the benefits brought by ENAS over a classic NAS algorithm, then the benefits of the two improvements introduced in sections 2.2.4 and 2.2.5.

### 5.3.1 Benchmarking ENAS vs NAS

In table 5.5 we see the average accuracy on the test set over 10 children networks sampled from the NAS and the ENAS controller, after 5 hours of training on 4 GPUs. The experiment was conducted on the ENAS version without the improvements discussed below. We notice how the ENAS algorithm allows for more iterations in the same amount of time, and a better mean accuracy compared to the classic NAS algorithm. The higher number of iterations performed by ENAS can be explained notably because of the action of an Early Stopping callback that generally stops the training earlier for children with layers that were already trained by previous iterations the ENAS algorithm.

On figure 5.1, we see the evolution of the evaluation accuracy during training (this graph is from a different training than in figure 5.5), we notice that although the ENAS architecture starts by sampling lower performance networks, it rapidly reaches the NAS performance with a higher line slope. [1]

[1] the experiment was conducted on 2000 iterations only, but after retesting on more iterations, the validation accuracy of the ENAS children went over the one of the NAS children

| | NAS | ENAS |
|---|---|---|
| Mean accuracy over 10 children | 0.72 | 0.75 |
| Amount of children sampled | 2514 | 3136 |

Table 5.5: Results of NAS vs ENAS

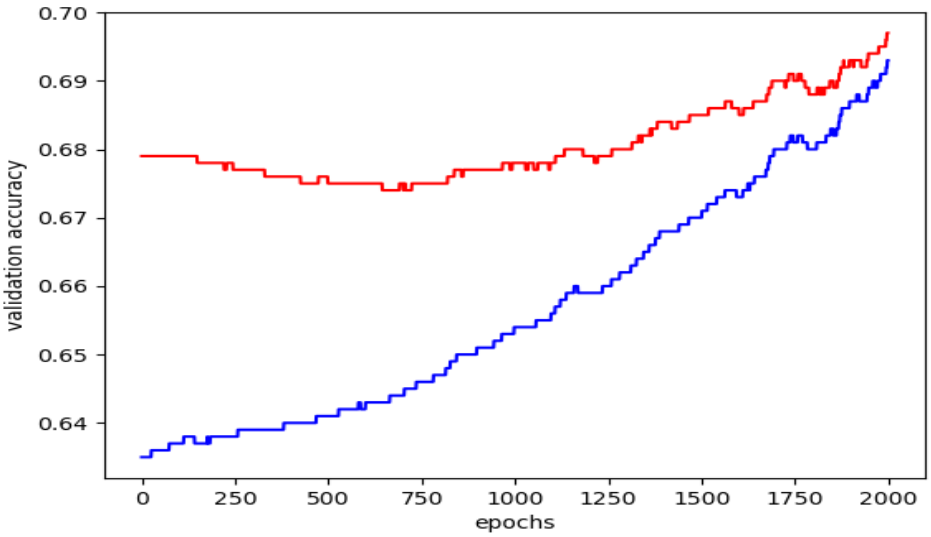
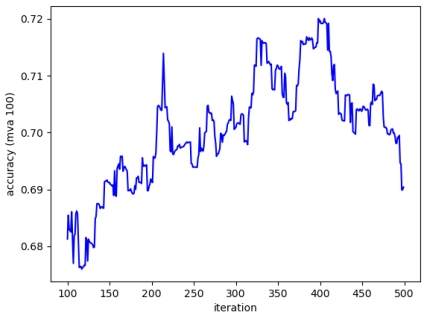



Figure 5.1: performances of NAS (red) vs ENAS (blue)

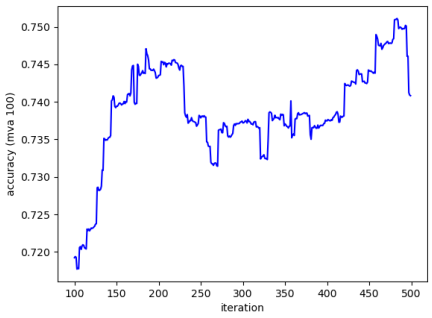
### 5.3.2 Benchmarking the two improvements on ENAS

In 5.2 one can see the benefits of the two additional features added to the ENAS algorithm (discussed in sections 2.2.4 and 2.2.5).

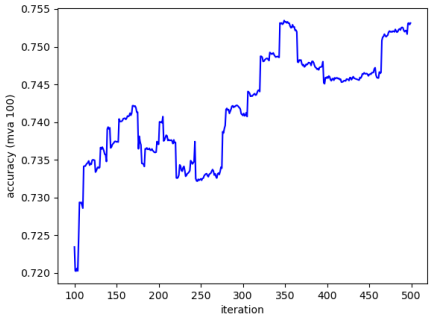
Using an adapted reward function and discarding worse accuracies both enforce the controller to sample even better architectures at each iteration. In particular, on 5.2c one notices that discarding worse accuracies allows the controller to rapidly resample good networks after a sudden collapse (it needs only about 100 iterations to recover from a bad sampling compared to 6.1b where around 200 iterations are needed).




(a) no adapted reward, no accuracy discarding




(b) adapted reward, no accuracy discarding




(c) adapted reward, accuracy discarding

Figure 5.2: Comparison of three differents versions of ENAS

### 5.3.3 Performances of the ENAS on each scheme

The ENAS algorithm augmented with the two previous improvements was tested over the four types of children architectures (schemes) defined in chapter 4. Each scheme is a different combination in term of the presence of skip connections and in term of the number of operations per block (1 or 2), table 5.6 shows for each combination the evaluation accuracy (moving average over 50 children) after 2000 iterations of the controller.

| # operations \ connections | skip | no skip |
|---|---|---|
| 1 | 0.78 | 0.84 |
| 2 | 0.78 | 0.82 |

Table 5.6: mva 50 of the evaluation accuracy after 2000 iterations

We can see, that the best accuracies come from scheme 1, i.e, with no skip-connections and one operation per block, the second best type of network being scheme 3, i.e, with no skip connections but two operations per block.

In tab 5.7 are shown the accuracies and Kappa scores on the test set for five children sampled from scheme 1 after a 10-fold cross validation training:

| | child 1 | child 2 | child 3 | child 4 | child 5 | mean |
|---|---|---|---|---|---|---|
| accuracy | 0.87 | 0.85 | 0.85 | 0.87 | 0.86 | 0.86 |
| Kappa | 0.75 | 0.72 | 0.72 | 0.75 | 0.74 | 0.74 |

Table 5.7: Results on scheme 1

For comparison in table 5.8 are the results of scheme 3.

| | child 1 | child 2 | child 3 | child 4 | child 5 | mean |
|---|---|---|---|---|---|---|
| accuracy | 0.80 | 0.84 | 0.83 | 0.79 | 0.87 | 0.82 |
| Kappa | 0.69 | 0.72 | 0.71 | 0.68 | 0.73 | 0.70 |

Table 5.8: Results on scheme 3

We see that the results are aligned with the evaluation accuracies shown during the training of the controller (table 5.6), i.e, that the ENAS controller that searched on scheme 1 samples better architectures than on other sub spaces.

## 5.4 Final results

From the previous section we determined that the best pair of Normal/Reduction cells were of the scheme 1, i.e, without skip connections and with only one operation per block. The $2^{nd}$ best scheme was found to be without skip connections and with two operations per block (scheme 3).

In a first section are shown the metrics computed after a 100-fold cross validation training on two architectures built after these two best schemes and in a second section are shown the most common blocks sampled for each of the four schemes.

### 5.4.1 Performances of architectures based on the two best schemes

After training, the controller sampled only a very limited number of the best architectures. For schemes 1 and 3 it turned out that a single architecture ended up emerging in an large majority (nearly 99% of the architectures were the same, see in next part).

Table 5.9 shows the best accuracy and kappa score (after post-processing) for each of the two schemes.

| | Scheme 1 | Scheme 3 |
|---|---|---|
| Best accuracy | 0.85 | 0.82 |
| Best Kappa score | 0.74 | 0.71 |

Table 5.9: Performances of the best architectures after a 100-fold cross validation

In appendix B the two corresponding architectures are show (figures B.8 and B.9).

### 5.4.2 Relevant design patterns

In this part are shown for each scheme, the most common blocks returned by the controller after training. Each block is numbered according to its position in the cell (Block 1 is the input of Block 2 for instance) [2] and between parenthesis is given the percentage of occurrence of the corresponding layer(s).

| | Block 1 (98) | Block 2 (97) | Block 3 (100) |
|---|---|---|---|
| Layer | Conv(2x5) | Conv(2x5_dl-4) | Conv(1x900) |

Table 5.10: Statistics for scheme 1, Normal cell

[2] for schemes with skip connections (2 and 4) there is always at east one connections (out of 2) between two consecutive layers

| | Block 1 (89) |
|---|---|
| Layer | Pool(1x6) |

Table 5.11: Statistics for scheme 1, Reduction cell

| | Block (89) 1 | Block 2 (88) | Block 3 | Block 4 (77) |
|---|---|---|---|---|
| Layer | Conv(2x10) | Conv(2x10_dl-4) | Conv(1x900) | Conv(2x5_dl-4) |

Table 5.12: Statistics for scheme 2, Normal cell

| | Block 1 (87) |
|---|---|
| Layer | Conv(1x6_strides-2) |

Table 5.13: Statistics for scheme 2, Reduction cell

| | Block 1 (95) | Block 2 (97) | Block 3 (97) | Block 4 (98) |
|---|---|---|---|---|
| Layer(s) | Conv(2x10) | Conv(2x10_dl-4)<br>Conv(8x1) | Conv(1x900)<br>Conv(2x5) | Conv(2x10_dl-4)<br>Conv(2x10) |

Table 5.14: Statistics for scheme 3, Normal cell

| | Block (100) |
|---|---|
| Layer | Conv(2x2_strides-2) |

Table 5.15: Statistics for scheme 3, Reduction cell

| | Block 1 (97) | Block 2 (100) | Block 3 (99) | Block 4 (100) |
|---|---|---|---|---|
| Layer(s) | Conv(2x10_dl-4)<br>Dropout(0.2) | Conv(8x1)<br>Conv(2x10) | Conv(1x900)<br>Conv(2x10_dl-4) | Dropout(0.2)<br>Conv(8x1) |

Table 5.16: Statistics for scheme 4, Normal cell

We can see that for all the schemes, the controller almost always samples the same blocks, for instance for scheme 3, the architecture built after the blocks in tables 5.14 and 5.15 is sampled 99% of the times (see also appendix B.9).

| | Block (99) |
|---|---|
| Layer | Conv(1x6_strides-2) |

Table 5.17: Statistics for scheme 4, Reduction cell

# Chapter 6

# Discussion

In this chapter, a first part discusses about the findings of the hyperparameter search over the Ansari CNN.

A second part discusses about the NAS algorithm efficiency, first by comparing the ENAS version with the classic one, then by comparing the ENAS augmented with the two new enhancements and without them, and finally by comparing the efficiency in the different search spaces (schemes).

A third part discusses the actual results of the ENAS algorithm in term of the architectures found.

A final part discusses the difficulties encountered during the thesis regarding the development and the use of the NAS algorithm and the relevance of the given solutions.

## 6.1 Hyperparameter search over the Ansari CNN

Two important results from the hyperparameter search over this base network emerged.

First, we noticed that the two best configurations showed a dilation rate of the $1^{st}$ convolutional layer equal to 4, this means that the combinations of pairs of electrodes that gave the best result is when the electrodes are distant from about the half of the total distance of the skull as represented in the 10-20 system (see figure 3.1), with no dilation rate, the first pair would be Fp1/Fp2 which are neighbouring electrodes whereas with a dilation rate of 4 the first pair would be Fp1/C4 which are more distant, this is also true for all the other pairs (the ordering of electrodes was the following: p1, Fp2, C3, C4, Cz, T3, T4, O1, and O2).

Second, for the four best results, one notices that they all display a larger number of filters compared to the original network (which has 10/20/20 layers respectively). This means that more features are to be detected in the data in order to obtain a better classification.

## 6.2 Results of the algorithm performances

### 6.2.1 Comparison of ENAS with classic NAS

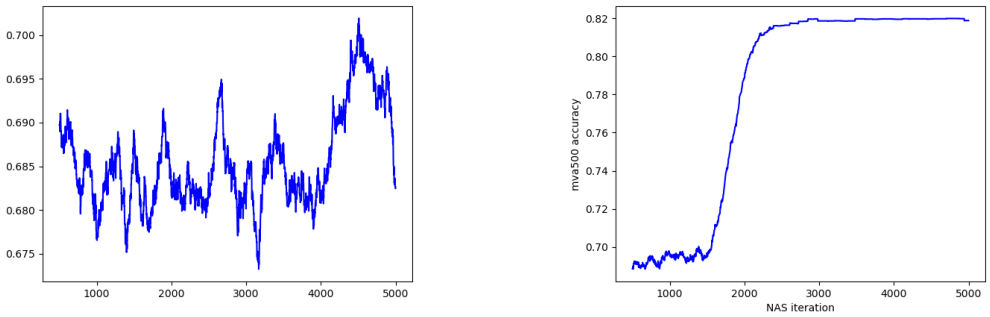
The weight sharing between children is the main principle that stands behind the ENAS improvement. Without weight sharing the different children do not take advantage of the training of the other children, which poses a problem in particular in NAS context, where the training of the children must be done quickly.

On figure 5.1, we see that the ENAS algorithm starts by sampling architectures that perfom less than the NAS, but we also see that it learns more rapidly and that the mean accuracy of its children soon reaches the mean accuracy of NAS children.

### 6.2.2 Performances of the two new enhancements

The new reward function and the improvement consisting of keeping only the best accuracies as a reward for a same architecture, both contributed to fine tune the behaviour of the NAS toward a more efficient training.

Below (figure 6.1) is another graph showing the importance of the new reward function, the experiment was performed without training of the children but by sampling the accuracies from a pre-computed distribution of the real accuracies. We clearly see the effect of the rewards normalization on the variance of the accuracies sampled.




(a) no adapted reward

(b) adapted reward

Figure 6.1: MVA 500 on sampled accuracies

### 6.2.3 Performance comparison between the four different schemes

The schemes with skip connections (2 and 4) correspond to the most intricate architectures and do not yield to results as good as the simpler schemes (1 and 3). This can be explained by two different factors. On one hand, the search subspaces themselves are maybe not as equally good in term of all potential children architectures to be found. On the other hand, the training time of more complex children slows down the training time of the controller, which leads, for the same

amount of training time of the controller to less sampled children and therefore to worse results.

## 6.3 Best architectures found and design principles

The performance results returned by the two best networks based on scheme 1 and scheme 3 (see 5.4.1) were found to be lower than of the Ansari network (0.74 for the architecture based on scheme 1 against 0.76 for Ansari). It is actually not a valid comparison, because the result from Ansari was based on a LOSO cross validation, which is much more accurate and generally leads to better results than the 100-fold cross validation that was performed in this thesis. Another explanation is that maybe the dataset does not allow to find better accuracies, notably because the size of the training set compared to the test set is almost equivalent, and that one would need much more training data to hope for a clear improvement.

The results from the controller sampling, and especially for scheme 1 (see B.8) confirm what was found during the hyperparameter search over the Ansari network, i.e, convolutional layers with two channels on the space axis and a dilation rate of 4 and with a timestep of 5 on the time axis are favoured compared to the initial parameters.

We can then make two conjectures regarding the design of a CNN consisting of stacked layers without skip connections (as for the Ansari network) adapted to the data of this thesis, i.e, a time window of 0.17 seconds (5 timesteps) of the EEG recording and a combination of electrodes that are distant from about half of the skull, both yield to better results in term of the CNN performance and therefore could also be taken into consideration for analysis by human experts.

In section 2.3.2 we discussed about the utility of clearly separating the extraction of features on the time axis on one side and on the channel axis on the other side (this was also experimented in [12]), to test this hypothesis two layers were introduced in the search space: Conv(8x1) and Conv(1x900). Both best architectures include Conv(1x900) and architecture from scheme 3 also includes Conv(8x1), this confirms that taking and extracting features separately on each axis also yield to better performances.

For schemes 2,3 and 4 we can see that the Reduction cell was constructed with a convolutional layer with a stride value of 2. This results confirm the best practice introduced in section 2.1.1. Indeed, it seems logical that by introducing more complexity, the networks learn better. This is particularly true for the NAS algorithm where the search is performed on small networks.

## 6.4 Development and use of the NAS

### 6.4.1 Difficulties

Adapting Keras to the requirements of a NAS algorithm can be challenging, for instance, there is no easy solution for retrieving layers outputs during the training of the controller. Other difficulties like running the code in parallel on multiple nodes/gpus was also challenging even-though Keras offers the corresponding software implementation.

Among the other challenges, one of the hardest to deal with was the *performance estimation strategy*, i.e, training the children represents the bottleneck of the NAS computational effiency and a trade-off between retrieving good estimations of the accuracies and shortening the training time as much as possible had to be found.

### 6.4.2 Solutions

Keras has the advantage to be relatively well documented and for most of the technical issues we can easily find answers on internet.

In this thesis, two major solutions were found to deal with the performance estimation strategy. The first one, was to evaluate the minimal number of epochs necessary to train each child and still retrieve an approximation of its relative value compared to the other children. The second one was the use of an adapted reward function to amplify the difference between children that are considered to be promising and the others and to reduce the high variance between rewards, which is a typical issue in a NAS algorithm. Both solutions proved to be leading toward better performances of the NAS. More importantly, the solution for the adapted reward function is also scalable and adaptive to any dataset, as its shape can be easily changed in order to adapt to the accuracies distribution relative to the different datasets.

# Chapter 7

# Conclusion

In this final chapter a conclusion of the thesis is drawn followed by suggestions for future work.

## 7.1 Conclusion

The crux of the thesis was to develop a NAS algorithm for sleep-stage classification for preterm infants. In the introductory chapter, we explained the background and the need to improve the classification algorithms of this type of EEG. Then, the second chapter enabled to detail the role of the NAS in automating the design of neural networks. Attention was paid in particular to the use of Reinforcement Learning as a learning agent that generates increasingly efficient architectures. The pseudo-code corresponding to the implementation of this algorithm was presented as well as new improvements, one already found in the literature, called ENAS, allowing to share the parameters in the generated children, and two others improvements, that were designed to increase the efficiency of the learning process, all of them were successfully implemented during this thesis. Then was presented the starting point of the thesis work, i.e the network developed by Ansari et al. A panomara of other papers also using CNN for this type of classification was drawn up. In chapter 3, an overview of what EEG are was given as well as the details regarding the dataset. In chapter 4, a concrete method was presented that showcased the importance of testing the NAS algorithm beforehand and presented a subdivision of the search space used.

With the hyperparameter search that was performed on the base network, a better configuration was found leading to an improved Kappa score of the network (0.78 instead of 0.76).

We saw that the NAS controller was able to sample increasingly better children networks and we clearly identified best design principles for CNN regarding the task for the binary classification of QS stages. Specifically, it was shown by both the hyperparameter search and the NAS results that another combination of channels (taken 2-by-2 but distant from half of the skull) and another timestep on the time

axis are two features that lead to increased accuracies. Finally, even if the ability of the algorithm to generate adapted architectures have been proven, it remains however, to test again the results by increasing the time and computational resources which were limited to 4 GPU during the thesis experiments.

## 7.2 Future Work

Several aspects can be considered to improve the work done during the thesis.

First, it was not possible to run the experiments on more than 4 GPUs at a time. This prevented, among other things, from carrying out a LOSO cross-validation in order to obtain results comparable with those of Ansari. After long discussions with the VSC technicians, we almost managed to get the algorithm work on several nodes (and therefore on more than 4 GPUs), and only a small effort is left to do in this direction.

The process of constructing and training the final architectures was inspired by [25] which was a suggested reference for this thesis. In this paper it is shown that stacking identical cells to each other allows, thanks to the principle of transferability, to obtain networks which generalize on more complex datasets. But, in the case of this thesis, only one dataset was used. Another idea to construct increasingly complex architectures is a technique called Progressive Neural Architecture Search (PNAS) [11], which constructs the children in a stepwise fashion, starting to test for children made of one block only, then adding in the search space of layers the layers of theses blocks that yield to the best performances, before eventually incrementing the size of the sampled children. With this technique the NAS algorithm directly constructs the final network by small steps. Intuitively we can think that the fact of stacking several identical cells to each other is suitable when we have the same type of data but in greater scale (typically when switching from CIFAR10 dataset to ImageNet dataset), PNAS, on the contrary, makes it possible to adapt a network by making it more complex while remaining on the same dataset.

As suggested in the reference paper on Reinforcement Learning for NAS [22], the evaluation of the children was based on their accuracy, as well as the reward that was used to train the controller. However, and even though a technique was developed to ensure the correlation between the accuracy and the kappa score (see section 4.2.2), it is this latter that was used as a reference to compare with the Ansari results. So it might have been better to use the Kappa score computed on the evaluation set to feed the reward function instead of using the evaluation accuracy. This should be part of a next experiment.

Regarding the data itself, and as it has been suggested by Ansari et al., a larger amount of data coming from several hospitals would obviously be very beneficial for the performance of the sampled CNN. Other data can be added to the input, for

example a spectral analysis of the EGG which would provide information on the frequency domain.

Finally, although the algorithm was developed in a modular and scalable way, it should be tested with a new choice of target architectures (other than CNN) and data (other than EEG).

# Appendices

# Appendix A

# Preparation step for the NAS algorithm

```
blocks_norm=3 # number of blocks per Normal cell
blocks_reduc=1 # number of blocks per Reduction cell
ops_conv=7 # number of layers available for the Normal cell
ops_reduc=4 # number of layers available for the Reduction cell
alts=1 # number of pairs Normal/Reduction cells to search

# instanciate the NAS Controller
controller = Controller(blocks_conv, blocks_reduc,ops_conv,ops_reduc,alts)

# plot a graph that shows accuracy & loss of 10 sampled arch over 20 epochs
controller.best_epoch(nber_sample=10, nber_epochs=20)
```

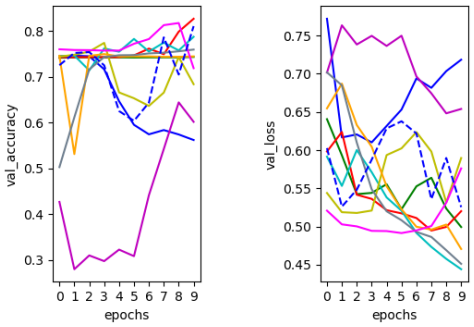



Figure A.1

```
# check for 10 sampled archs whether the reward (here 'val_accuracy')
# is correlated with the final metrics (here 'kappa score')
controller.match_train_test_metrics("val_accuracy", "kappa_score", 5)
```

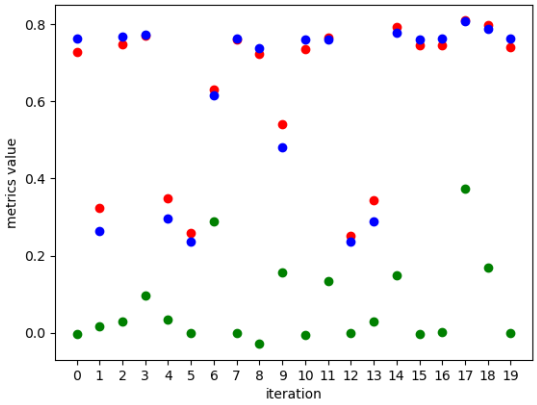



Figure A.2: train vs test metrics

```
# create a file with the 'training' accuracies computed on
# 50 sampled children
controller_instance.compute_accuracies(50, 2, batch_size, print_file=0)
```

```
# test how many iterations are needed for the NAS to increase the mean
controller.test_nb_iterations( file_accs = 'accuracies.txt', limit=3000)
```

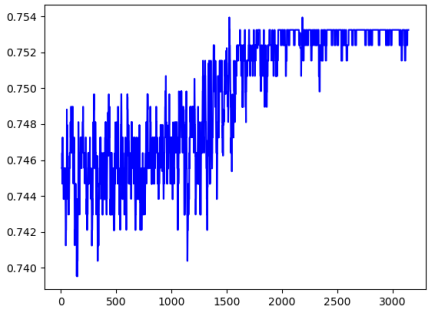



Figure A.3

We can see that after around 1700 iterations the algorithm reaches a plateau where it generates networks with highest accuracies.

```
# Finally, train the NAS algorithm
global_batch_size = 256x4 # if 4 GPUs for instance
nb_child_epochs=2
mva1=10
mva2=500
epochs=5000

params = {
    'global_batch_size' : 256x4,
    'nb_child_epochs' : 2,
    'mva1' : 10,
    'mva2' : 500,
    'epochs' : 5000
}

controller_instance.train(params)
```

This above line generates multiple files of data, we can for instance see the movering average of the accuracy over 500 iterations.

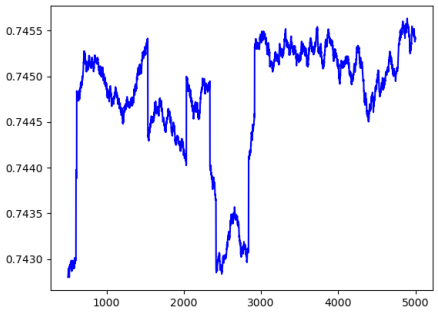



Figure A.4

We notice that the increasing of the accuracies is not as regular and as high compared to the graph returned by the "test_nb_iterations" function. This is because the new reward function was not yet implemented for this experiment.

# Appendix B

# Visual representation of the architectures

## B.1 Normal cell examples for each of the four schemes

### B.1.1 Scheme 1

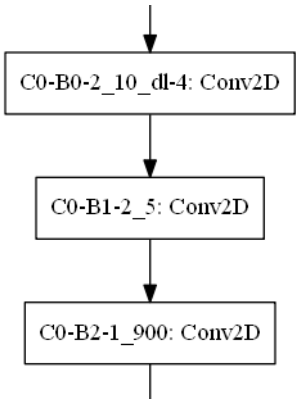



Figure B.1: no skip-connections, 1 operation per block (3 blocks)

### B.1.2 Scheme 2

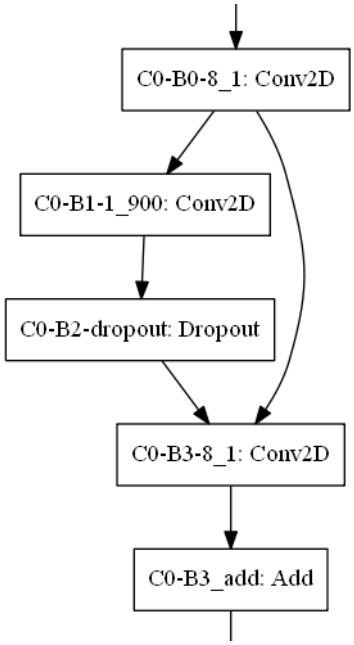



Figure B.2: skip-connections, 1 operation per block (4 blocks)

### B.1.3 Scheme 3

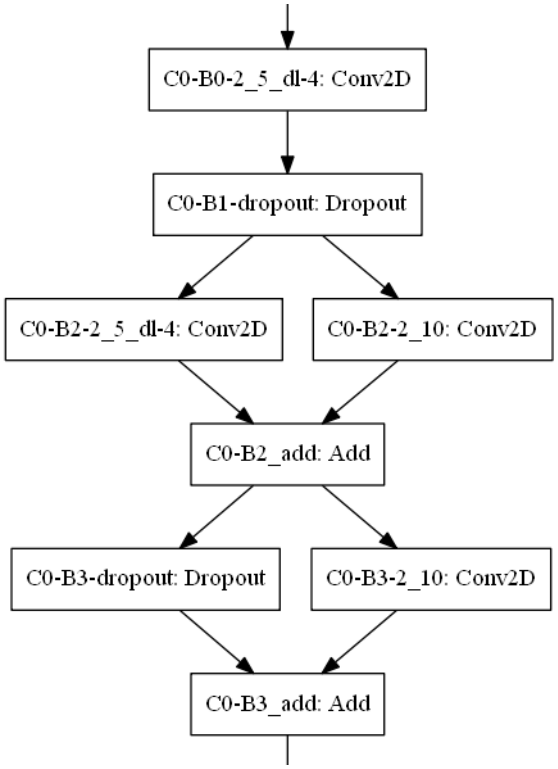



Figure B.3: no skip-connections, 2 operations per block (4 blocks)

### B.1.4 Scheme 4

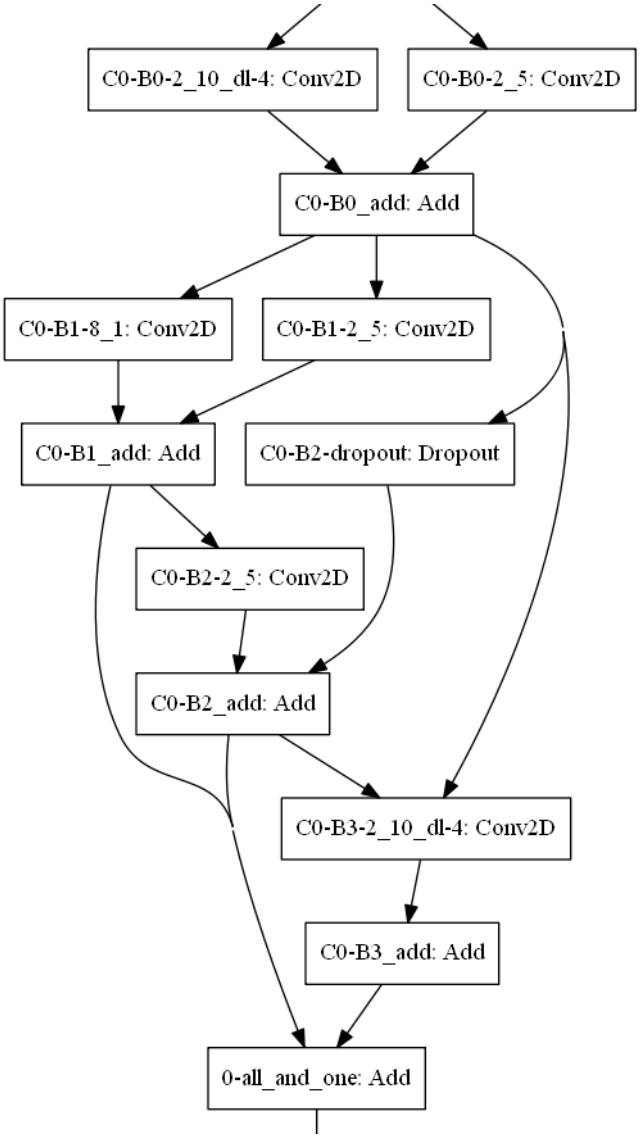



Figure B.4: skip-connections, 2 operations per block (4 blocks)

## B.2 Reduction cell

### B.2.1 Reduction cell

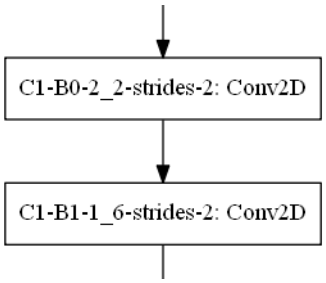



Figure B.5: Reduction cell

## B.3 Examples of complete architectures

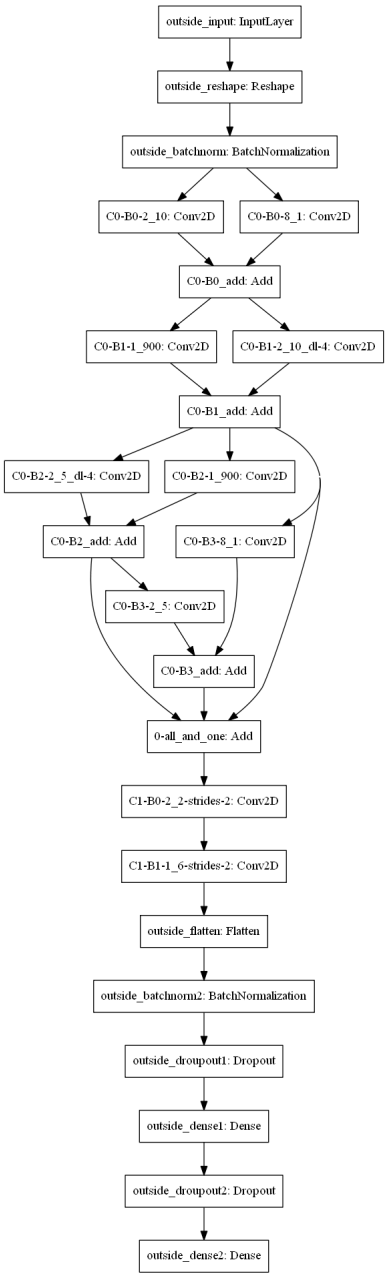



Figure B.6: Network with one Normal cell (C0) followed by one Reduction cell (C1)

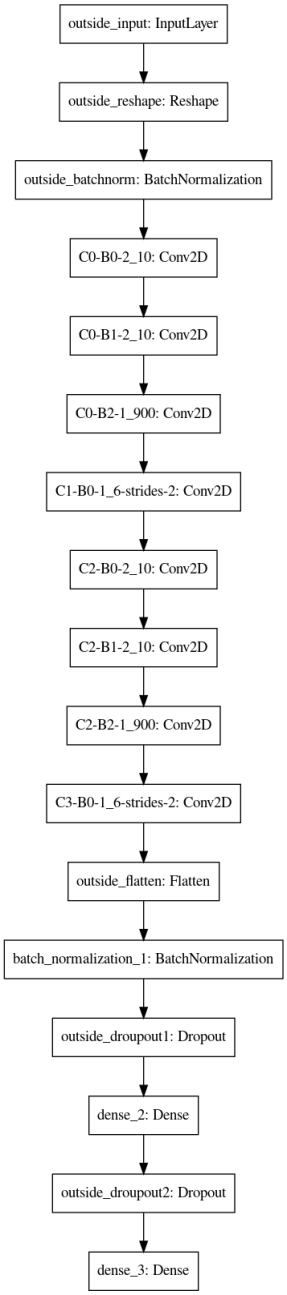



Figure B.7: Example of a final network, stack of 2 pairs (C0/C1 and C2/C3)

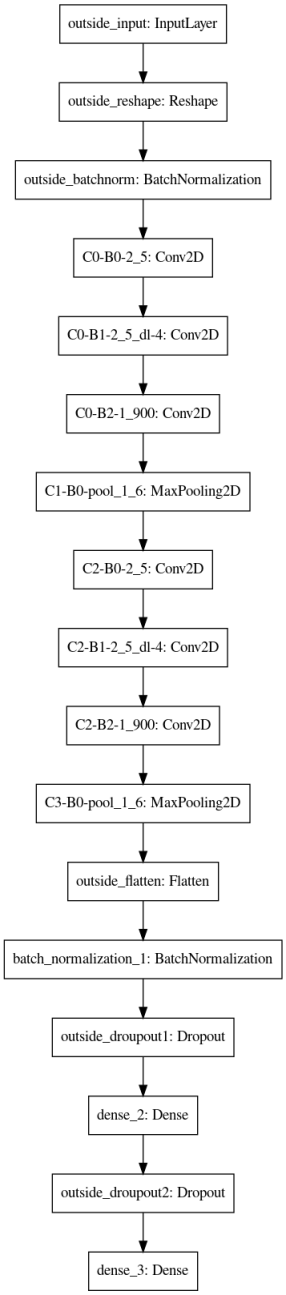



Figure B.8: Architecture made after scheme1, result from the controller after training

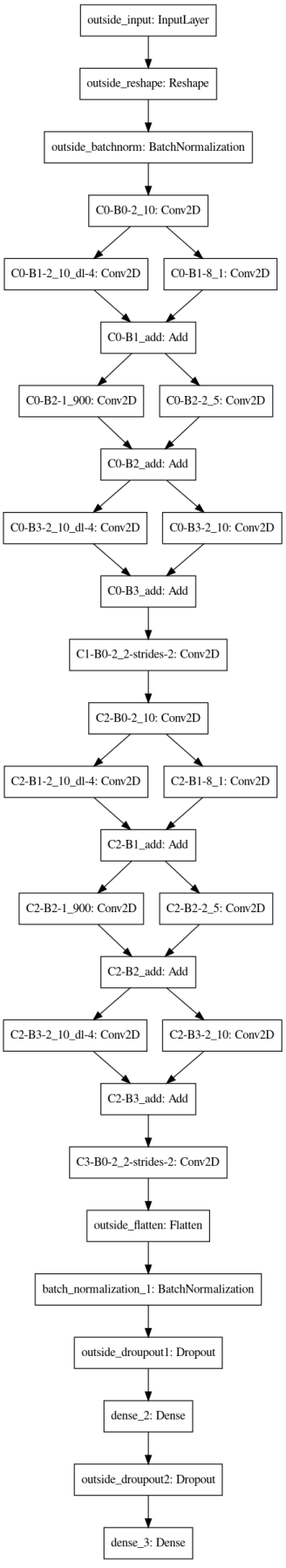



Figure B.9: Architecture made after scheme3, result from the controller after training

# Bibliography


[1] 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.

[2] A. H. Ansari, O. De Wel, K. Pillay, A. Dereymaeker, K. Jansen, S. Van Huffel, G. Naulaers, and M. De Vos. A convolutional neural network outperforming state-of-the-art sleep staging algorithms for both preterm and term infants. *Journal of neural engineering*, 17(1):016028, 2020.

[3] C. Brunner, M. Billinger, C. Neuper, and G. Pfurtscheller. Influence of spatial filters on the performance of eeg-based bcis using autoregressive models. In *Neuromath Workshop 2009*, pages 37–37. ., 2009.

[4] F. Chollet et al. Keras, 2015.

[5] A. Dereymaeker, K. Pillay, J. Vervisch, S. Van Huffel, G. Naulaers, K. Jansen, and M. De Vos. An automated quiet sleep detection approach in preterm infants as a gateway to assess brain maturation. *International journal of neural systems*, 27(06):1750023, 2017.

[6] Y. Gao, B. Gao, Q. Chen, J. Liu, and Y. Zhang. Deep convolutional neural network-based epileptic electroencephalogram (eeg) signal classification. *Frontiers in Neurology*, 11:375, 2020.

[7] A. A. Garvey and E. M. Dempsey. Applications of near infrared spectroscopy in the neonate. *Current opinion in pediatrics*, 30(2):209–215, 2018.

[8] S. Graven. Sleep and brain development. *Clinics in perinatology*, 33(3):693–706, 2006.

[9] R. W. Homan. The 10-20 electrode system and cerebral location. *American Journal of EEG Technology*, 28(4):269–279, 1988.

[10] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. 4(4), 1990.

[11] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[12] X. Lun, Z. Yu, T. Chen, F. Wang, and Y. Hou. A simplified cnn classification method for mi-eeg via the electrode pairs signals. *Frontiers in Human Neuroscience*, 14:338, 2020.

[13] W.-L. Mao, H. I. K. Fathurrahman, Y. Lee, and T. W. Chang. EEG dataset classification using CNN method. *Journal of Physics: Conference Series*, 1456:012017, jan 2020.

[14] K. Palmu, T. Kirjavainen, S. Stjerna, T. Salokivi, and S. Vanhatalo. Sleep wake cycling in early preterm infants: comparison of polysomnographic recordings with a novel eeg-based index. *Clinical Neurophysiology*, 124(9):1807–1814, 2013.

[15] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018.

[16] A. Piryatinska, G. Terdik, W. A. Woyczynski, K. A. Loparo, M. S. Scher, and A. Zlotnik. Automated detection of neonate eeg sleep stages. *Computer methods and programs in biomedicine*, 95(1):31–46, 2009.

[17] L. Torrey and J. Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[18] Wikipedia contributors. Dilution (neural networks) — Wikipedia, the free encyclopedia, 2020. [Online; accessed 2-June-2021].

[19] Wikipedia contributors. Automated machine learning — Wikipedia, the free encyclopedia, 2021. [Online; accessed 28-May-2021].

[20] Wikipedia contributors. Eeg analysis — Wikipedia, the free encyclopedia, 2021. [Online; accessed 31-May-2021].

[21] Wikipedia contributors. Rectifier (neural networks) — Wikipedia, the free encyclopedia, 2021. [Online; accessed 2-June-2021].

[22] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[23] C. Zhang, Y.-K. Kim, and A. Eskandarian. Eeg-inception: An accurate and robust end-to-end neural network for eeg-based motor imagery classification, 2021.

[24] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[25] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable n. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.