



ÉCOLE CENTRALE LYON

UE PRO
PAR 133
REPORT

AI for Integrated Electronic Circuit Diagnostics

Students :

Baptiste GRIGNON
Neirouz BOUCHAIRA

Supervisors :

Emmanuel DELLANDREA
Alberto BOSIO

Project advisor:

Denis MAZUYER

May 2023

Abstract :

This research project focuses on the application of artificial intelligence methods for fault diagnosis of integrated electronic circuits. In particular, it is about the application of machine learning models based on convolutional neural networks. The industrial objective behind the research project is to be able to trace the position and nature of the fault, from a series of simple tests on a faulty circuit, in particular if the fault does not manifest itself at each test. In this case, we call it an "intermittent fault". The presence of intermittent faults makes the diagnosis much more difficult and they are not taken into account by the current industrial tools. In this study, we propose a fault diagnosis method that includes intermittent faults, based on convolutional neural networks. We then evaluate the performances of those models in different cases, including against the standard industrial diagnosis tool.

Contents

Summaries and acknowledgements	1
1 Introduction	3
1.1 Context	3
1.2 Project Objectives	4
2 State of the art	5
2.1 Electronic circuit faults	5
2.1.1 Fault model	5
2.1.2 Permanent faults and intermittent faults	6
2.1.3 Grouping into metaclasses	7
2.2 Neural Network Architecture	9
2.2.1 Deep Neural Networks (DNNs)	9
2.2.2 Convolutional Neural Network (CNN)	10
2.2.3 Vocabulary	10
3 Approach	11
3.1 Study circuit and simulation	11
3.2 Database Creation	11
3.3 Database formatting by hierarchical clustering	13
3.3.1 Hierarchical ascending clustering (HAC)	13
3.3.2 Database formatting	13
3.4 Neural network models	14
3.4.1 Convolutional neural network used	14
3.4.2 Hierarchical approach	15
4 Experiments and results	16
4.1 Trained models	16
4.2 Performance of models compared to the training activation rate	18
4.3 Performance of models faced with new activation rates	20
4.4 Performance Improvement by Using an Activation Rate Distribution	21
4.5 Diagnostic test: CNN model versus industrial tool	21
5 Conclusion	21
References	23
Annexes	24

1 Introduction

1.1 Context

Production of electronic circuits

Every year, hundreds of billions of electronic circuits are produced for applications in all fields. The manufacturing of an integrated circuit begins on a wafer (figure 1). These are disks made of semiconductor material (Silicon in most cases) on which the circuits will be etched.

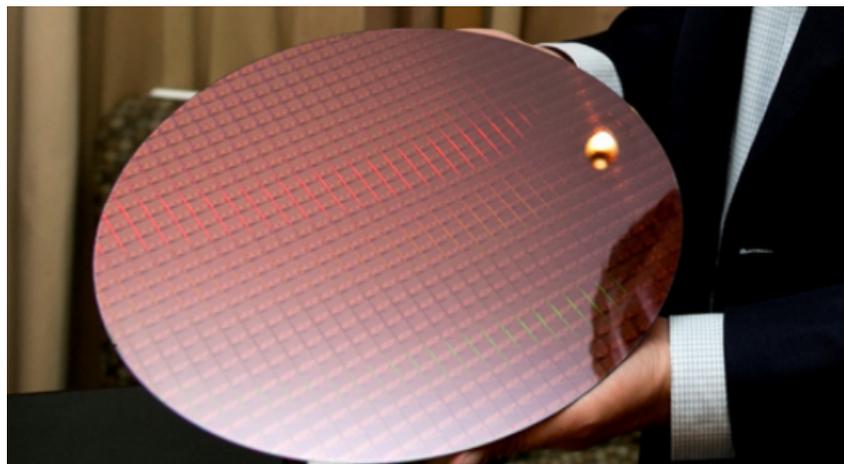


Figure 1: Wafer at the end of manufacturing

Test phase

When the manufacturing of the circuits of a wafer is finished, they are all tested using a pin called the Wafer probe. Indeed, the appearance of errors in the production of printed circuits remains frequent. However, it is necessary to be able to guarantee the quality and reliability of the components produced and to get rid of the faulty ones. This is the stage that our application project focuses on.

Challenges

Our application project focuses on the diagnosis of these faulty circuits. It involves precisely identifying the error present on a faulty electronic circuit. To do this, we apply different inputs to the circuit and record the outputs. Industrial tools already make it possible to trace the origin of the error using these tests and the structure of the electronic circuit. However, industrial tools are not always reliable because they do not allow us to identify errors that do not systematically appear in all tests. We are talking about intermittent faults.

The challenge of the project is therefore to examine whether artificial intelligence methods can be used to increase the accuracy of diagnostics and take intermittent faults into account.

1.2 Project Objectives

Previous Projects

This application project is in its third iteration. The previous group [5] has succeeded in showing that methods based on deep neural networks and in particular on convolutional neural networks (also called CNN, for "Convolutional Neural Network") can be interesting for the diagnosis of electronic circuits. Indeed, after having divided all possible faults into groups, they managed to predict with a high success rate to which group of faults any tested error belongs.

Objectives

The general objective of the project is to continue the research already started by the previous group on the application of CNNs to the diagnosis of electronic circuits. In particular, it is a question of establishing and testing a method to more precisely identify an error within the groups established by the previous research group, potentially up to finding the fault itself.

In addition, the difficulty in identifying an intermittent fault depends on a parameter defined in the state of the art section, called the activation rate. However, the neural networks used for identification must be trained with an artificially created database presenting a chosen activation rate. However, the activation rate of the errors that these networks will be confronted with is not necessarily known in advance. It is therefore also a question of establishing methods for training networks to improve their robustness with respect to a diversity of activation rates.

Finally, it would be interesting to compare the performance of the networks that we will have trained with the existing industrial tool through the diagnosis of any intermittent fault.

2 State of the art

2.1 Electronic circuit faults

Sources: [5] and the explanations of Dr. BOSIO.

2.1.1 Fault model

Stuck at model

We model electronic circuit faults using a so-called "stuck at" model. The principle is to consider that faults can be located on the inputs and outputs of each component of the circuit and that each of these wires can be "stuck at 0" or "stuck at 1", i.e. the Boolean value present on the wire is stuck at the value 0 or at the value 1, regardless of the operation of the rest of the circuit. The set of stuck at 0 and stuck at 1 wires constitute the set of possible faults.

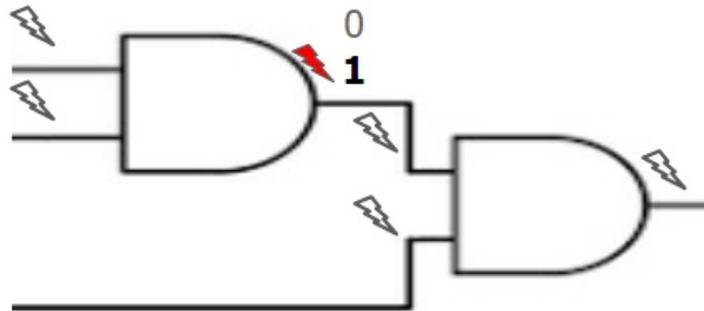


Figure 2: Example of applying the stuck at model

Let's take an example using the figure 2. Each lightning represents a potential fault location. It can be noted that 2 faults of the same nature ("stuck" to the same Boolean value), present on the same wire are inseparable. We then speak of equivalent faults.

Fault matrix

We define a test vector as a vector of Boolean numbers to be applied at the input of the real circuit in order to perform the diagnosis. For the example circuit (figure 2), the potential test vectors are:

n°	Test vector
1	(0,0,0)
2	(0,0,1)
3	(0,1,0)
4	(0,1,1)
5	(1,0,0)
6	(1,0,1)
7	(1,1,0)
8	(1,1,1)

For larger circuits (with more inputs), the number of potential test vectors explodes. In these cases, we choose a reasonable number among all the possible test vectors, so as

to maximize the number of faults that this set of vectors can reveal. We consider that a test vector can reveal a fault when, when we apply the vector as input, the output of the circuit presenting the fault in question is different from the output of the non-faulty circuit.

For each test vector applied to the circuit, we obtain a row vector of Boolean values corresponding to the outputs of the circuit. Thus, after having applied to the circuit the set of test vectors chosen previously, we obtain a **matrix** whose **rows** correspond to the **test vectors**, and whose **columns** correspond to the **circuit outputs**. This matrix is the **characteristic** of the circuit that we use for its diagnosis. A circuit will have a different matrix depending on the fault it presents, or whether it is non-faulty. By extension, we will speak of the matrix of the fault "X" (where X is its identifier) and of a non-faulty matrix. Furthermore, we will consider for the moment that **a circuit can never present more than one fault** at a time.

If we take the example circuit (figure 2), to which we apply vectors n°1, 3 and 6, we obtain for the stuck at 1 fault at the location in red the following matrices:

$$\left\{ \begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right\} \qquad \left\{ \begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right\}$$

Fault Matrix Non-Fault Matrix

Figure 3: Matrices of the example circuit for the fault in red

It may seem strange to speak of a matrix when they only have one column, but this is because the circuit taken as an example has only one output: the matrices have as many columns as the circuit has outputs. Furthermore, an error is identified based on 2 things:

- its position on the circuit
- its polarity: stuck at 0 or stuck at 1

Finally, we can say that **circuit diagnosis is the process of tracing the fault position and its polarity from the matrix resulting from the application of the test vectors to the circuit**. The matrix constitutes the "signature" of the fault present on the circuit.

2.1.2 Permanent faults and intermittent faults

Some faults do not appear each time a test vector is applied, which makes them difficult to diagnose. We speak of "intermittent faults", as opposed to "permanent faults" which are always present.

We can try to characterize an intermittent fault by its activation rate T_a . For an intermittent fault:

$$T_a = \frac{\text{number of test vectors that activate the fault}}{\text{number of test vectors applied}} \quad (1)$$

A vector "activates" the fault when it highlights it. If we refer to figure 3, only test vectors 3 and 6 can highlight the fault, i.e. only lines 2 and 3 are different between the fault matrix and the non-faulty matrix. We then have 3 possible scenarios:

- $T_a = 100\%$, the 2 test vectors activate the fault, the fault matrix is the one presented in figure 3;
- $T_a = 50\%$, only one of the 2 vectors activates the fault, the fault matrix can be one of the 2 presented in figure 4;
- $T_a = 0\%$, this is the case of the absence of faults, the matrix is the non-faulty matrix.

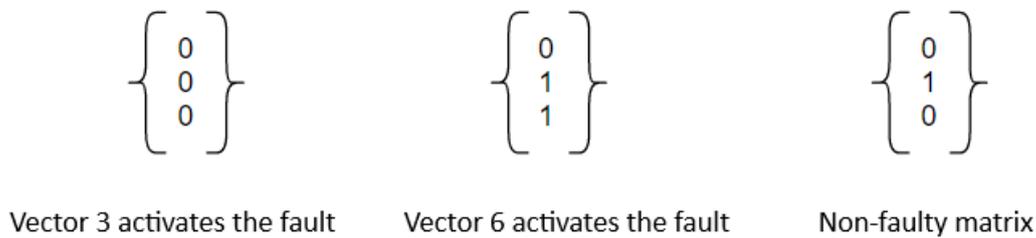


Figure 4: Faulty matrices of the example circuit for the fault in red on the example circuit, with $T_a = 50\%$

Here, we have given a simple example but when the number of test vectors is larger, the number of possible combinations explodes, even if we know the activation rate (which is not the case in general).

The current approach used by industrial tools is to list the matrices of all permanent faults and compare them to the matrix obtained as a result of the test until a match is found. However, if the fault present on the tested circuit is intermittent, this induces variations in the matrix obtained as a result of the test compared to the matrix of the corresponding permanent fault that we are trying to find. The diagnostic tool will therefore not find a match between the matrices and the diagnosis will fail. Furthermore, the lower the activation rate, the more the intermittent fault matrix is different from the intermittent fault matrix and therefore the more difficult the diagnosis is.

This observation justifies the study of other diagnostic methods to take these intermittent faults into account. In particular, methods based on neural networks have generalization capabilities that could allow them to diagnose intermittent faults.

2.1.3 Grouping into metaclasses

Definitions:

Generally speaking in machine learning, a **classification problem** consists of **finding the category of an object based on information** that it carries. For example, we have images of animals and 2 categories: cats and dogs, and we seek from the RGB matrix of the image to identify which images present cats and which images present dogs. We

call a **category** (cat or dog in the example), a **class**. The problem we are dealing with is also a classification problem but the classes are the fault identifiers and the information is the matrix resulting from the circuit test. We define a **metaclass** as a **class that designates a grouping of permanent faults**. We can define metaclasses to **reduce the number of classes that the neural network must classify**. Indeed, a circuit can potentially have thousands or even millions of different permanent faults and it would be far too costly in terms of computing resources (or even impossible) to train a neural network that directly classifies the fault. We therefore initially just classify the metaclass.

Construction of metaclasses by the previous research group

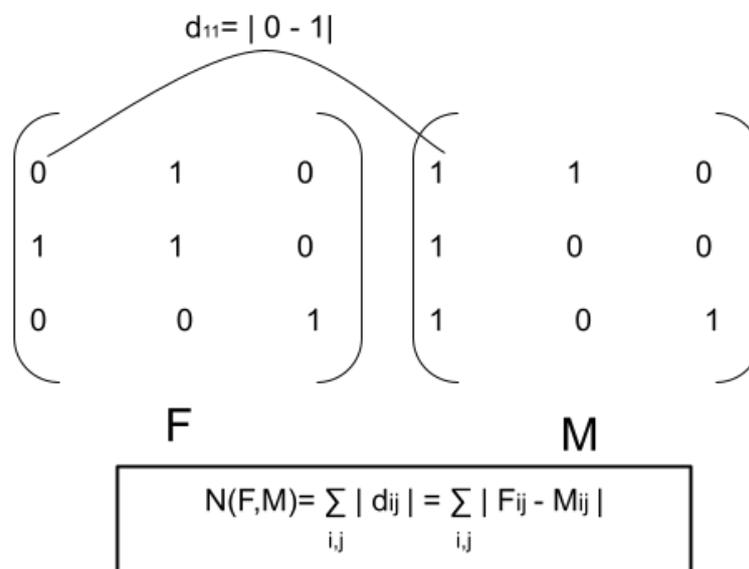


Figure 5: Definition of the distance for the metaclasses of the previous research group on this subject

The method of grouping faults into metaclasses, used by the previous research group on this subject and based on **the norm 1** of the difference between the matrices of permanent faults, allowed them to group the 3609 possible different faults for the c7552 circuit (which is the study circuit, see approach section) into 121 metaclasses. The 1-norm of the difference is calculated as the sum of the differences between elements of the same position in the two matrices to be compared, as shown in Figure ???. The matrices with the smallest norms were placed in the same metaclass, and all metaclasses contained approximately equal numbers of faults.

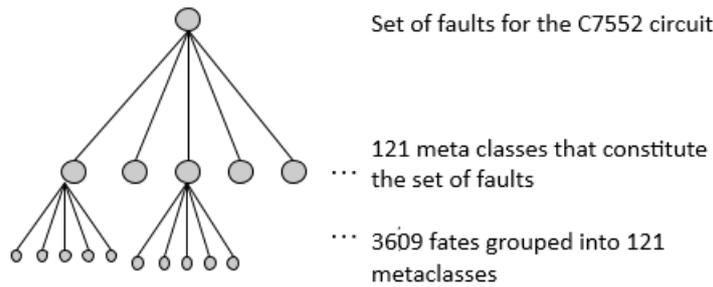


Figure 6: Classification of faults by stage for the c7552 circuit

Hierarchical method:

The previous research group proceeded to a grouping of the classes to have a total of 121 classes instead of 3609. Then, they carried out the training of the model which identifies the meta class of the fault. The classification stopped at the level of the meta classes (stage 1 of the figure 6) and there was no downstream classification work to determine the fault more precisely.

However, they posed a **hierarchical method** which consists for a matrix whose fault we want to determine to first determine the meta class using a "main" model. Then, using "sub-models" (one for each meta class) that also take the fault matrix as input, but this time we know the meta class, we classify the fault precisely within this meta class. In this way, we reduce the problem to a branch of the tree of possibilities before going to the end of the resolution. This method replaces the use of a single model with an absurd number of parameters with the use of a larger number of models, but each of which has a reasonable number of parameters. We will strongly take up this concept (see approach section).

2.2 Neural Network Architecture

Sources: [2], [4], and [6] Artificial intelligence (AI) is a branch of computer science that aims to create machines that can replicate some human cognitive functions such as perception, natural language understanding, learning, and problem solving. Machine learning is a subdiscipline of AI that allows machines to learn without being explicitly programmed. Machine learning algorithms can be trained on large amounts of data to detect patterns and establish relationships between inputs and outputs. Deep learning is a branch of machine learning that uses artificial neural networks with multiple layers to perform complex tasks, such as image recognition, language translation, and speech recognition.

2.2.1 Deep Neural Networks (DNNs)

Deep neural networks (DNNs) are computational models inspired by the structure of biological neural networks. They consist of multiple layers of neurons connected together, where each layer performs computations on the input data to produce more complex features.

Traditional neural networks are typically shallow, with one or two hidden layers. DNNs, on the other hand, typically have many hidden layers, sometimes up to several

dozen. This additional depth allows DNNs to capture more complex features in the input data.

DNNs are typically trained using the **backpropagation** algorithm, which involves adjusting the network weights to minimize the difference between the network's predictions and the target values (the cost function). DNNs typically require considerable training data to produce accurate results.

2.2.2 Convolutional Neural Network (CNN)

A convolutional neural network (CNN) is a type of neural network that is particularly effective for image recognition. It consists of several layers of neurons, each with the task of detecting features of the image, such as lines, angles, textures, etc.

The **operation** of a CNN is based on the use of *convolutional filters*, which are mathematical patterns applied to the image to detect specific features. Each layer of the network uses a number of filters to analyze the image and extract relevant information.

The CNN also uses *max-pooling* layers, which reduce the size of the image while retaining the most important features. This simplifies the analysis of the image by the following layers of the network and reduces the computation time required.

Finally, the CNN is trained using matrices and their corresponding label (e.g. the fault matrix and the label 'fault-name'). The network then modifies its weights and biases to minimize the prediction error. Once trained, the CNN can be used to predict the label of new matrices.

2.2.3 Vocabulary

Metric: a measure that quantifies the performance, quality, or efficiency of a system, algorithm, or application.

Accuracy: is a performance measure used in machine learning to evaluate the accuracy of a classification model. It is calculated by dividing the number of correct predictions by the total number of predictions.

3 Approach

3.1 Study circuit and simulation

For our study, we use a circuit model provided by the tutors and whose name is *c7552*. It is relatively small in terms of the number of logic gates, has 108 outputs, and a series of 55 test vectors allows us to highlight the 3609 permanent faults that it can present. An example of a fault matrix is given in figure 7:

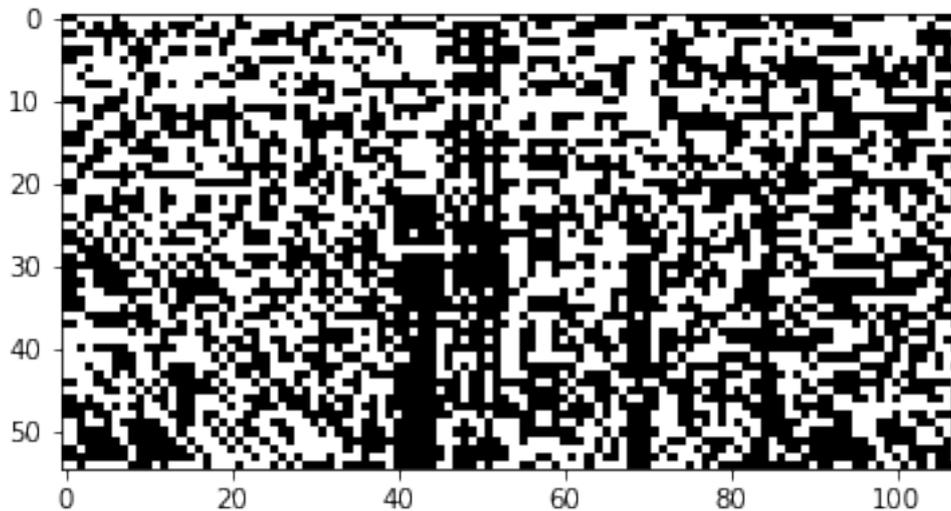


Figure 7: Example of a fault matrix (0 in white, 1 in black)

This representation highlights the similarity between the matrix of a tested circuit and a black and white image, which justifies the use of convolutional neural networks, mainly used to recognize patterns in images.

It is then important to note that we do not have the real circuit and that this study is conducted entirely using simulation. Indeed, we have on the INL servers the industrial software **Tetramax** for which we have a model of the *c7552* circuit. This is the tool that allows us to find all the permanent faults as well as the list of 55 test vectors. It then allows us to simulate the circuit outputs for each of the permanent faults and therefore to determine all the permanent fault matrices.

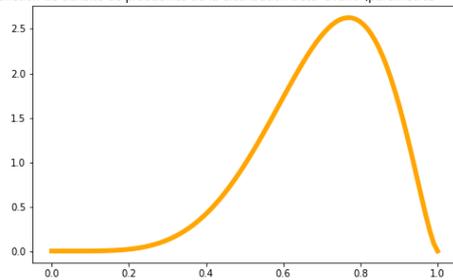
3.2 Database Creation

We previously discussed the fact that the number of possibilities for intermittent fault matrices is exploding. The advantage of using a neural network model is that it can be trained on a database containing only a sample of these possibilities and if the sample is sufficiently varied, the model generalizes to other examples on which it has not been trained. Therefore, it is vital to judiciously create the database on which to train our model (in fact, we will create several databases to train several models).

To do this, we introduce a new parameter n , which is the number of samples. This is the number of matrices of intermittent faults corresponding to the same permanent fault present in the database. A database will therefore always contain $3609 * n$ matrices.

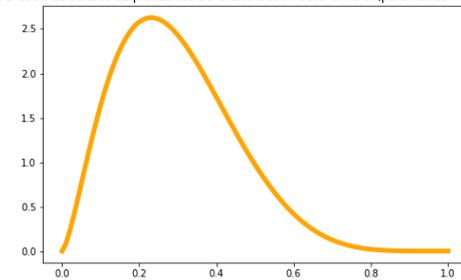
In addition, we must choose the activation rate that the intermittent fault matrices present in the database. The previous research group only explored the possibility of an identical activation rate for the entire database. We will further explore the use of activation rates chosen differently for each matrix, following a certain probability distribution. However, we lack information on the probability distribution of the activation rates of intermittent faults in real circuits. We therefore assume that since the manufacturing techniques and tools are uniform for a sample of a circuit, the intermittent faults will present similar characteristics, and therefore the activation rates will be roughly grouped. This consideration led us to use the following distributions (beta distribution for different parameters):

Fonction de densité de probabilité de la distribution beta 'avant' (paramètres = (6, 2.5))



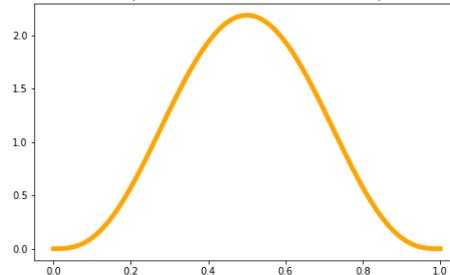
(a) Probability density of the distribution that we will call "beta forward" with parameters= 6, 2, 5 (original French figure)

Fonction de densité de probabilité de la distribution beta 'arriere' (paramètres = (2.5, 6))



(b) Probability density of the distribution that we will call "beta rear" with parameters= 2, 5, 6 (original French figure)

Fonction de densité de probabilité de la distribution beta 'milieu' (paramètres = (4, 4))



(c) Probability density of the distribution that we will call "middle beta" with parameters= 4, 4 (original French figure)

Note, however, that even for a database with a fixed activation rate, the product $T_a \cdot \text{number of vectors potentially activating the fault}$ is rarely an integer. For each matrix, the integer part of this product gives us the number of vectors that will activate the fault (which we then randomly choose for each matrix with a draw without replacement following a uniform distribution). Consequently, even for a database with a fixed rate, we will observe small variations in the activation rate between faults.

Finally, we randomly mix the database and divide it into 3 parts: 60% which are used for training, 20% which are used for validation (i.e., to verify that the model does not over-specialize for the recognition of training data, inducing worse performances outside of these data), and 20% which are used to test the model after training.

3.3 Database formatting by hierarchical clustering

3.3.1 Hierarchical ascending clustering (HAC)

Hierarchical ascending clustering is a data classification method that organizes observations into successive groups, thus forming a hierarchy of groups, hence its name. It is based on measures of similarity or distance between observations, and groups those that are most similar or closest to each other. The groups formed are then used to create larger groups, until the complete hierarchy of groups is formed.

In order to obtain different metaclasses, it is possible to cut the clustering tree at a given distance. This method is often used in hierarchical clustering to obtain a given number of groups or classes. This makes it possible to simplify the visualization of the tree and facilitate the interpretation of the results. To do this, it is first necessary to construct a dendrogram from the distances calculated between the different groups. Then, it is necessary to identify the height corresponding to the desired distance to obtain the groups at this distance.

The groups obtained following this cut are subsequently called metaclasses. We can then proceed to lower cuts giving us other groupings included in the previous ones. The metaclasses thus group several **subclasses**, which can be considered as subgroups of faults. The hierarchical classification thus makes it possible to structure the fault data into several levels of grouping, from individual faults to metaclasses, via subclasses.

We will then use the hierarchical method mentioned in the state of the art section for the classification of faults: we start by using a first network to determine the metaclass of the fault. Then, we have one network per metaclass, and we ask the one of the metaclass that we determined in the previous step to determine the subclass of the fault and so on until we go down to the identity of the fault.

However, this requires training a very large number of networks and it is not really necessary for the tests so we will stop at the prediction of the first level of subclasses (which for our division is equivalent to the identity of the fault, as we will see later).

3.3.2 Database formatting

The Database formatting is done using the CAH, using the norm 1 of the difference as a similarity criterion like the previous research group. That is to say that we make a copy of the database where the identifiers of the faults are changed by the identifiers of the metaclasses, and another copy that we subdivide between each metaclass and for which the identifiers of the faults are changed by the identifiers of the subclasses (as many sub-databases as metaclasses).

The difference with the work of the previous research group lies in 2 things. The first lies in the fact that our metaclasses do not each contain a roughly identical number of faults but that the number of faults per metaclass (and per subclass) depends on the number of "children" of its branch after the division. The interest of this method is that when we seek to discriminate between metaclasses or subclasses at a given level, the differences between them are roughly equivalent. The second lies in the fact that the previous research group had no concept of subclasses. Indeed, we realized that to have reasonable performances, these intermediate groups are necessary.

To clarify these concepts, here is the dendrogram of the permanent fault matrices of the c7552 circuit:

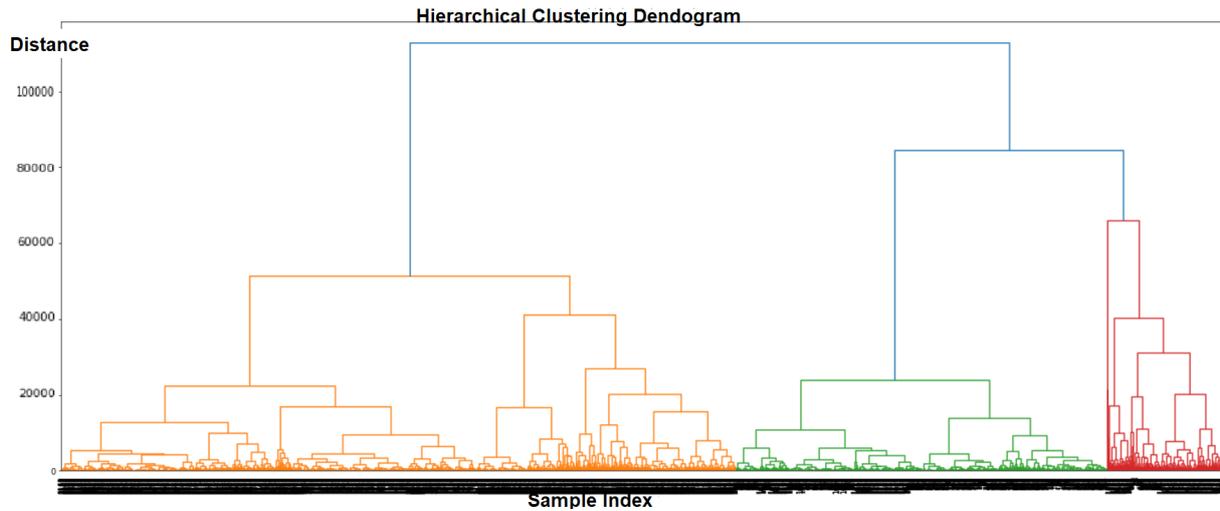


Figure 9: Example of a dendrogram obtained by CAH

We can see it as a tree whose leaves are the faults and whose nodes are the successive groupings of 2 groups (the elementary groups being the leaves). The ordinate of the nodes is the distance between the groups at the time of their grouping.

For our databases, the distance at which we cut the dendrogram to form the meta-classes is 13000, which gives us 20 meta-classes. The distance at which we cut to form the subclasses is 700, which gives us 603 subclasses distributed between the different meta-classes. Most of these subclasses are clusters of less than 10 faults and a good number are clusters containing only one fault. We therefore consider that for the tests there is no need to go further with a new level of subclasses. The number of subclasses per meta-class and faults per subclass for our clustering is available in the appendix.

3.4 Neural network models

3.4.1 Convolutional neural network used

CNN architecture

The convolutional neural network used to predict the meta-classes (figure 10) is composed of several layers, each with a specific functionality to extract the relevant features from the input fault matrices.

The first layer is a **Conv2D** layer with 128 filters and a ReLU activation function. This layer convolves the input image with a set of learned filters to extract important features from the image. Next, a **MaxPooling2D** layer with a pooling size of 2x2 is used to reduce the dimension of the Conv2D layer's output. This layer preserves the most important features of the image while reducing the amount of data. They are followed by a second Conv2D layer with 64 filters and a MaxPooling2D layer.

A Flatten layer is then used to transform the output of the last Conv2D layer into a one-dimensional vector. This vector is used as input for the fully-connected layers that follow. The first 2 of these fully-connected layers (Dense) have 64 neurons and use ReLU activation. These layers allow the neural network to combine the features extracted from the Conv2D layers to produce higher-level features.

The regularization function 'dropout' is applied with a regularization rate of 0.2 in the neural network model. This means that at each training iteration, 20 % of the neurons of

the previous layer are randomly chosen and ignored during this iteration, which helps to avoid overfitting.

Finally, the last Dense layer has a number of neurons equal to the number of classes we are trying to identify. This layer uses a softmax activation function to produce a probability for each possible class, which allows us to determine the final class to which the input data belongs by choosing the one with the highest probability.

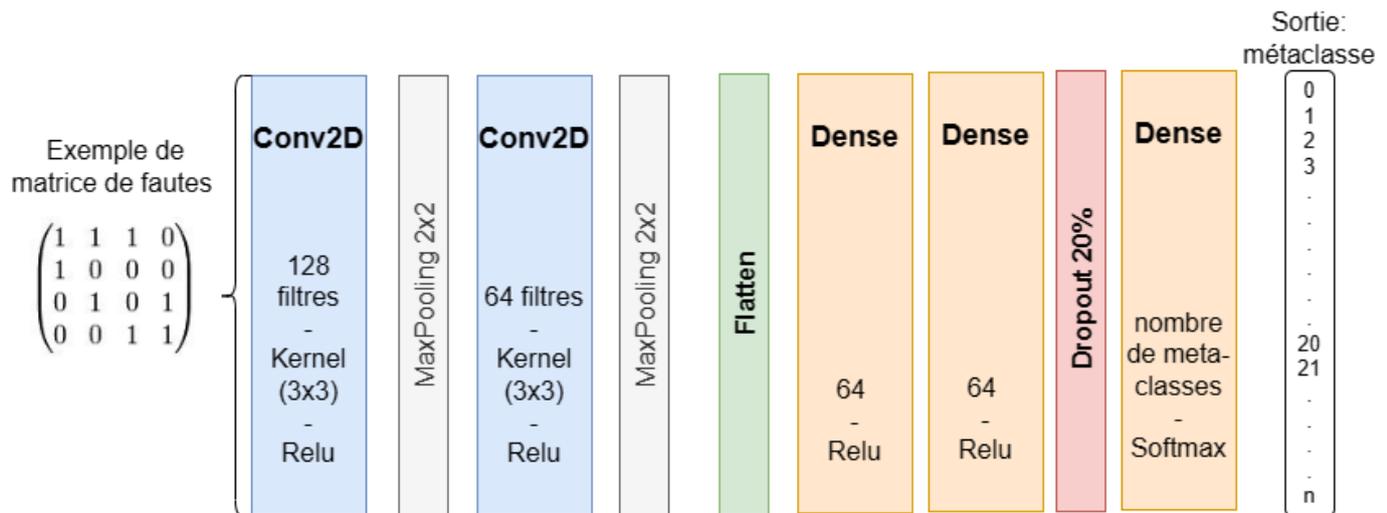


Figure 10: CNN architecture for meta-class classification (original French figure)

Model Training

In this model, the Stochastic Gradient Descent (SGD) optimizer is used to minimize the 'sparse-categorical-crossentropy' loss function. The SGD optimizer works by adjusting the model weights and biases using the gradients of the cost function with respect to these parameters. The 'sparse-categorical-crossentropy' loss function is used because it is suitable for multi-class classification tasks.

A metric for evaluating the model's performance is specified: the accuracy. The accuracy is the proportion of correct predictions out of the total number of predictions.

Finally, we define the number of "epochs", that is to say the number of iterations: predictions, calculation of the loss function, then correction of the parameters by back-propagation, on 150. In addition, at each iteration, all the training data are used.

3.4.2 Hierarchical approach

We train a first network whose structure is as presented previously to recognize the meta-classes using the formatted database presented previously.

Then, it is a question of training the networks allowing to determine the subclass within the meta-class (one network per meta-class). These networks also have the same structure as the "main" network. At first we hoped to train them by transfer learning, that is to say copy the parameters of the "main" network on the "subnetworks" (which classifies the subclasses), and re-train them using the "subdatabases" formatted previously by modifying only the parameters of the last layer, the classification layer. This approach gave bad results, so in the end we trained all the parameters of these networks from 0.

Once these networks are trained, we can for any faulty matrix, predict its meta-class, then its subclass and thus know a fairly restricted group of faults that the tested circuit

having given the matrix in question is likely to present. We can call this prediction process using multiple models with different parameters, a **metamodel**.

4 Experiments and results

4.1 Trained models

Taux d'activation	Caractéristique β	n
Constant	10%	30
	30%	30
	50%	30
	70%	30
	80%	30
	80%	10
	80%	5
	90%	30
Distribution	β arrière	30
	β avant	30
	β milieu	30

Figure 11: Databases used for training (for the beta distribution, arrière: rear, avant: forward and milieu: middle) (original French figure)

We used a "metamodel" introduced in the previous research group graph. This model was trained on several different databases, each of them characterized by different activation rates and numbers of samples (n) (figure 11). These characteristics were used to create eleven distinct "metamodels", which differ by the specificities of the databases on which they were trained.

The cost function is what the algorithm seeks to minimize during training. Here, it is the cross-entropy function, which measures the difference between the predicted probabilities and the true probabilities for each class. This cost function is widely used for classification tasks and allows to measure how well the model's predictions match the training data. We also used the accuracy metric to measure the precision of our models.

Example database: $T_a = 30 \%$, $n = 10$

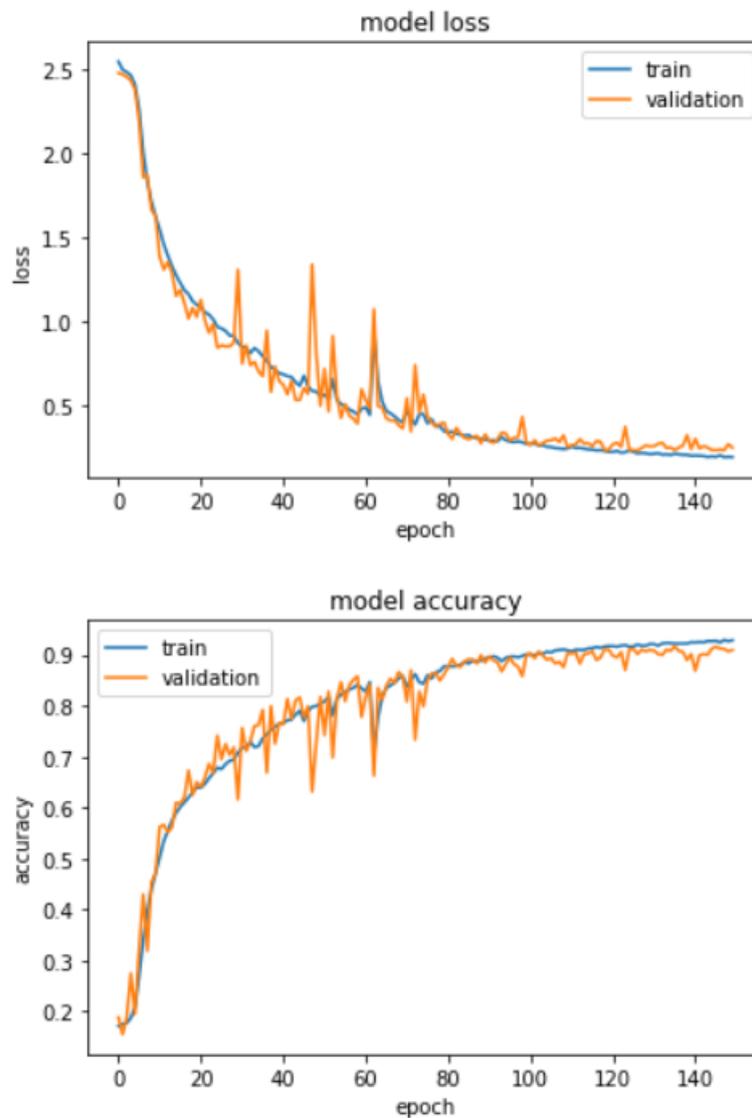


Figure 12: Model cost and accuracy function for metaclass classification, database: $T_a = 30 \%$, $n = 10$

For this example (12), the training and validation cost function converge to a value of about 0.2, which suggests that our model has reached an optimal solution for the neural network parameters. Similarly, we observed that the training and validation accuracy converged to a value of about 0.92, which indicates that our model is able to generalize well on new data.

The validation curve is not used directly during training but to check that there is no overfitting. Indeed, we evaluate the model metrics on the training part of the database and on the validation part during the entire training. We adjust the parameters using the cost function calculated on the training part, the validation part has no influence. On the other hand, if the training and validation curves move away from each other, this indicates that there is overfitting: the model is too specialized in recognizing training data and loses its generalization capabilities.

To differentiate between the accuracy of a model and the accuracy of a metamodel, some figures will mention the "total accuracy". This is also the number of successful predictions divided by the total number of predictions, but for the entire prediction process of the metaclass and then the subclass.

4.2 Performance of models compared to the training activation rate

First, we examine the performance of the metamodels on the "own" data sets. That is to say that for each of the metamodels, we calculate the total accuracy from the part dedicated to the tests of the database that was used to train them. The results are presented in figure 13:

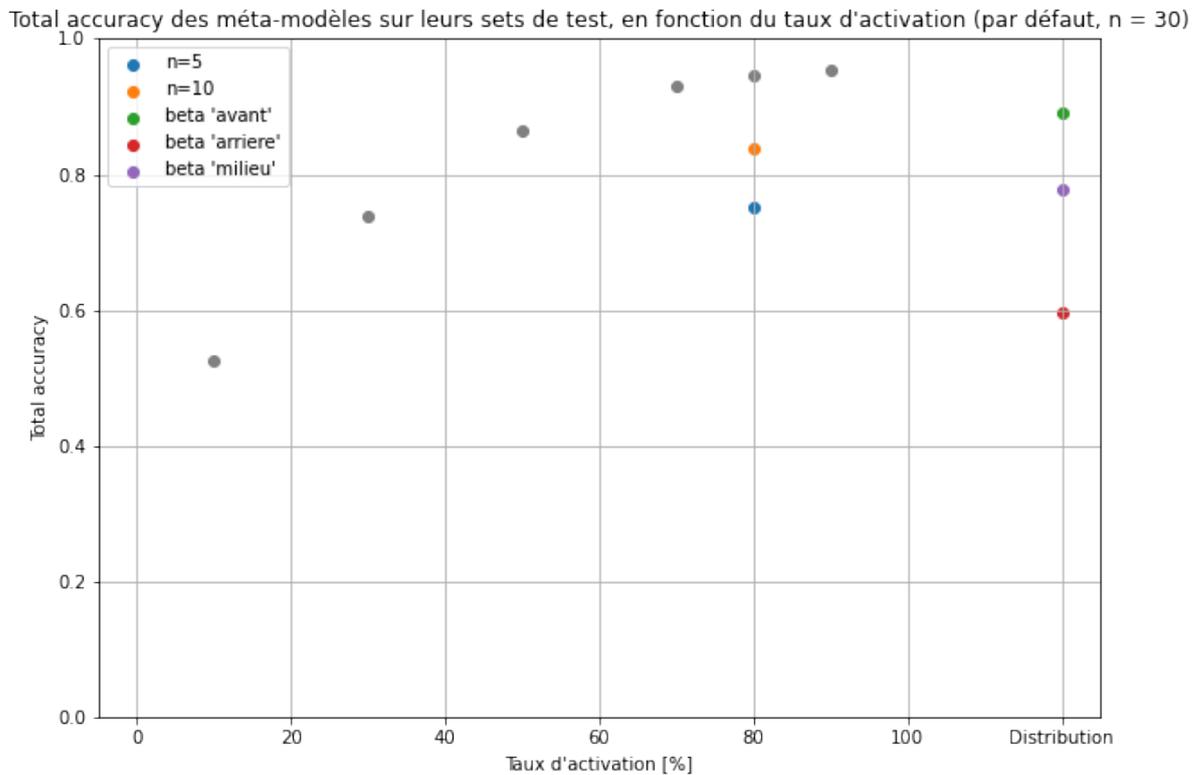


Figure 13: Accuracy of metamodels on their own test data as a function of the activation rate (for the beta distribution, arriere: rear, avant: forward and milieu: middle) (original French figure)

We can notice 2 things:

- at an equal number of samples, metamodels trained at fixed rate have a higher total accuracy than models trained when the activation rate increases. This can be explained because for a lower activation rate, the number of possibilities for the fault matrix increases, and with the same number of samples, we cover a smaller part of these possibilities.

- at an equal activation rate, the greater the number of samples, the greater the total accuracy. We can explain it with the same logic: the number of possibilities for the fault matrix is the same, but with fewer samples we cover fewer possibilities.

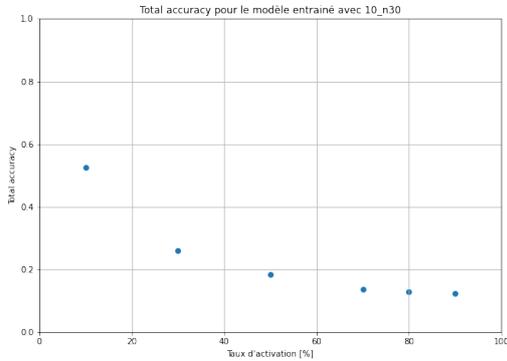
We can conclude that to obtain the best results **we must maximize the number of samples**. However, this can be difficult because we often encounter memory problems. To give an example, a database containing $n = 30$ samples weighs less than 5 GB, so the RAM allocated to the training program must be at least 5 GB. By increasing the number of samples, we can quickly run out of memory. With the resource at our disposal, it was not possible for us, for example, to train a network with a database at $n = 30$.

The activation rate of real circuits is not known a priori; however, these results show that the activation rate of a fault represents the difficulty in diagnosing the fault: the lower it is, the higher the difficulty.

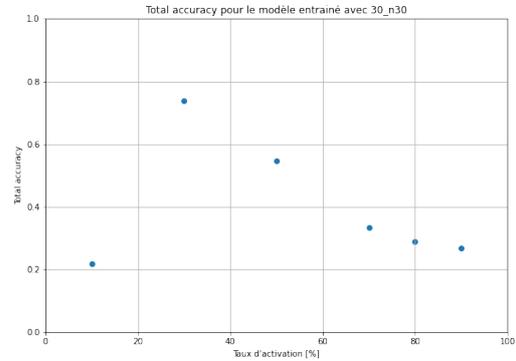
For databases containing a wider spectrum of activation rates (distributions called "beta forward", "backward" and "middle", or even approach part), we make a similar observation: the larger the point around which the distribution is centered, the greater the total accuracy.

Finally, we still note that the results are rather encouraging: the minimum total accuracy is greater than 50% and the maximum total accuracy exceeds 95%. However, these metrics are calculated with data that have the same activation rate as the metamodel. In reality, this would be equivalent to knowing the activation rate in advance, which is not the case. The question is therefore the following: How do metamodels perform when faced with databases with activation rates different from those on which they were trained?

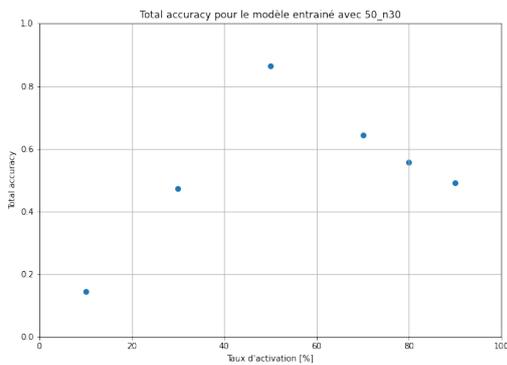
4.3 Performance of models faced with new activation rates



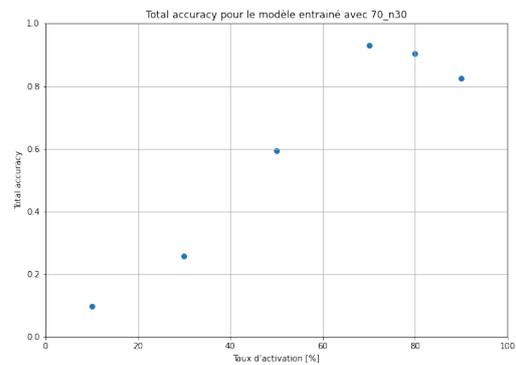
(a) Total accuracy for a model trained with an activation rate of 10 % (n = 30)



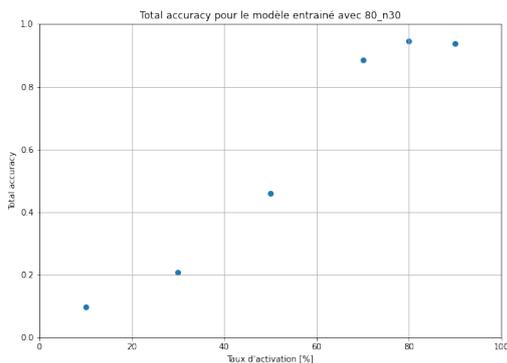
(b) Total accuracy for a model trained with an activation rate of 30 % (n = 30)



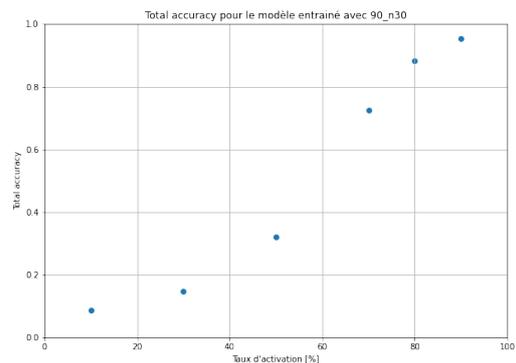
(c) Total accuracy for a model trained with an activation rate of 50 % (n = 30)



(d) Total accuracy for a model trained with an activation rate of 70 % (n = 30)



(e) Total accuracy for a model trained with an activation rate of 80 % (n = 30)



(f) Total accuracy for a model trained with an activation rate of 90 % (n = 30)

In a second step, we compare the different trained metamodels to the test databases of the other metamodels: we then obtain the total accuracy of the metamodels as a function of the activation rate from the test database.

We notice that generally, the graphs present a "peak" at the level of the activation rate of training of the model, a decrease to the left of the peak, and a slower decrease to

the right of the peak (we see it particularly well on the figure 14c).

We can say that the larger the area under the curve, the more efficient the model is, because the more it will be able to have high accuracy over a wider range of activation rates. In this sense, we can say that metamodels trained at activation rates between 50

4.4 Performance Improvement by Using an Activation Rate Distribution

We first note that the area under the curve for these metamodels is significantly larger than those trained at fixed rates, so we can say that metamodels trained with fixed rates distributions are generally more efficient than the others. We also notice that the shape described by the curves changes: we no longer have a peak but instead we have a plateau on the right and a regular growth on the left. The more the distribution is centered towards the right, the higher the plateau and the steeper the part.

Finally, we conclude that using a distribution for activation rates does indeed improve diagnostic performance, and that in particular a distribution centered around 50

4.5 Diagnostic test: CNN model versus industrial tool

Finally, we still have to compare the best of our metamodels (the one trained on an activation rate following the "beta middle" distribution) with the industrial tool. The diagnosis by the tool is heavy and being at the end of the project we were only able to test it for a single fault (more experiments on this aspect would be necessary) chosen at random: the stuck at 0 fault on the U1446/Z wire. Moreover, this fault is the only one in its subclasses so our metamodel can potentially identify it precisely.

To begin with, the industrial tool and the metamodel find the fault from the permanent matrix, which is expected. Then, we generate an intermittent version of the fault matrix where only 6 test vectors out of the 21 potential ones activate the fault, i.e. the fault has an activation rate of about 30

This result is nevertheless to be qualified because if the fault was not alone in its subclass, we could only have restricted the possibilities to faults that are members of this subclass without having been able to identify it precisely (during a test, we could however know if we have identified the right fault group or not).

5 Conclusion

In conclusion, during this project we developed a method for diagnosing intermittent faults in electronic circuits, which industrial tools do not yet take into account. To do this, we grouped the faults using a hierarchical ascending classification method, from which we deduced two levels of fault grouping: metaclasses that contain subclasses, subclasses that contain faults. We then trained a battery of neural networks to determine the metaclass and then the subclass of a fault from its matrix.

Finally, we studied the performance of these models based on different parameters, which allowed us to conclude that training the models with an activation rate distribution centered around 50 We have also shown that our model could successfully diagnose an intermittent fault, where the current industrial tool fails to do so. This result, although currently limited to a single test, is encouraging for the future. In the future, it would be

necessary to go all the way with the approach, by ensuring that the subclasses of the last level only present one fault, for precise identification. It could also be interesting to look at the case where the circuit presents several faults, or to perform tests with a physical circuit.

References

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep Networks with Stochastic Depth. 2016. Publisher: arXiv Version Number: 3.
- [3] NVIDIA. Cuda toolkit documentation. Disponible à <https://developer.nvidia.com/cuda-toolkit> (06/04/2023).
- [4] Josh Patterson and Adam Gibson. *Deep learning en action: la référence du praticien*. First interactive O'Reilly, Paris, 2018.
- [5] Kévin-lâm Quesnel and Nantomiaro Ralibera. PAR 135 : IA appliquée au diagnostic de circuits électroniques intégrés. Technical report, April 2022.
- [6] Bharath Ramsundar and Reza Bosagh Zadeh. *TensorFlow pour le deep learning: de la régression linéaire à l'apprentissage par renforcement*. First interactive O'Reilly, Paris, 2018.

Annexes

Project management

In this research project, project management was a key element in achieving our objectives. We found that despite the initial use of the Gantt chart to plan our tasks, it was important to remain flexible and adaptable, especially given the evolution of research tracks. Indeed, we have encountered new questions since RVP2, such as the use of constant activation rate databases or distributions for training our networks, as well as the addition of the classification track using Transfer Learning. These questions required adjustments in our schedules and a reorganization of our work. Despite these changes, we managed to move forward with our experiments, although we were somewhat behind schedule due to the creation of the datasets taking longer than expected. Project management was therefore crucial to address these challenges and achieve our main objective, which is to study the application of AI for the diagnosis of electronic circuits.

GANTT

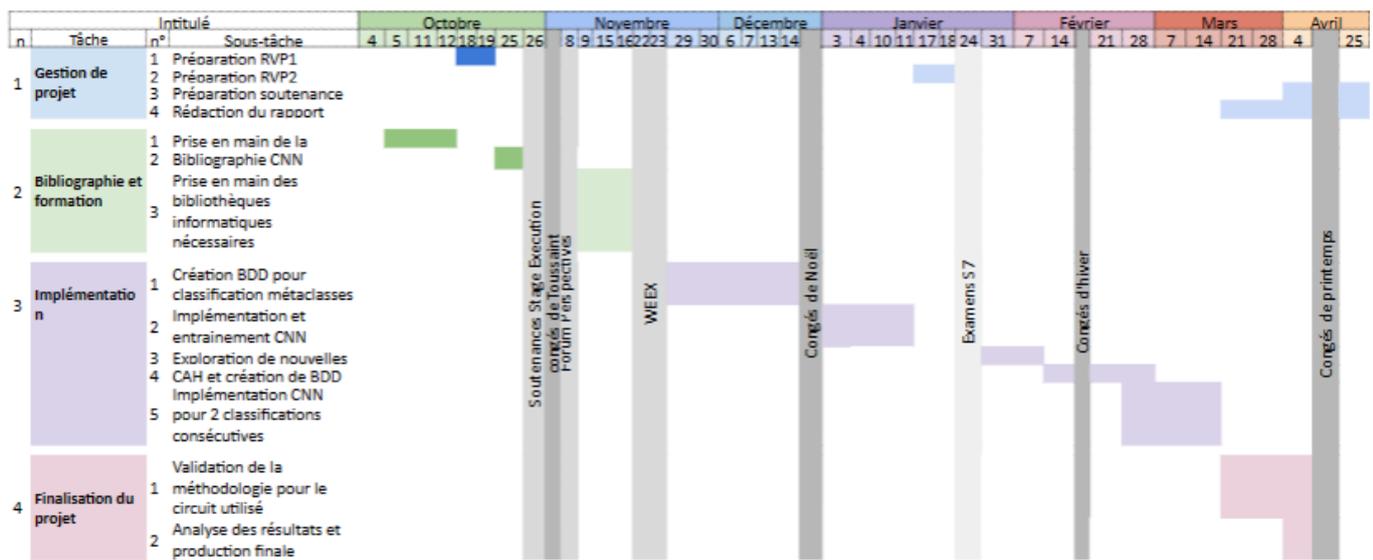


Figure 16: GANTT final (French original figure)

The final Gantt (figure 16) is a visual representation of our updated project schedule. It shows the various tasks that have been completed. Our project’s final Gantt chart includes several key tasks, including bibliography and training, implementation, and project completion. The Gantt chart was updated regularly throughout the project to reflect changes in priorities and timelines.

Changes from the last review with supervisors

The implementation subtasks were revised, adding an exploration of new paths based on RVP2 feedback to find improvements over the previous research group. Once the tracks were found and validated by the professors, we worked on hierarchical ascending classification (HAC), two-stage classification and analysis of the results obtained.

The subtask of generalization to other circuits in the project finalization task was finally abandoned. We chose to focus on the track of 2 consecutive classifications and left aside the generalization to other circuits.

Code

The codes used are all available on the project gitlab: <https://gitlab.ec-lyon.fr/diagnostic-de-circuits-electroniques/par-133>. Only the databases are not on the gitlab because they are too massive. The library used for neural networks is tensorflow [1]. The acceleration of matrix calculations by using NVIDIA GPUs is performed by [3].

Distribution of faults in metaclasses into subclasses

Size for metaclasses corresponds to the number of subclasses inside the metaclass. Size for subclasses corresponds to the number of faults inside the subclass.

Metaclass 0, size 59:

- Subclass 0, size 14
- Subclass 1, size 11
- Subclass 2, size 23
- Subclass 3, size 10
- Subclass 4, size 11
- Subclass 5, size 11
- Subclass 6, size 34
- Subclass 7, size 13
- Subclass 8, size 8
- Subclass 9, size 36
- Subclass 10, size 16
- Subclass 11, size 7
- Subclass 12, size 14
- Subclass 13, size 17
- Subclass 14, size 42
- Subclass 15, size 33
- Subclass 16, size 14
- Subclass 17, size 52
- Subclass 18, size 15

- Subclass 19, size 12
- Subclass 20, size 15
- Subclass 21, size 7
- Subclass 22, size 7
- Subclass 23, size 3
- Subclass 24, size 6
- Subclass 25, size 12
- Subclass 26, size 13
- Subclass 27, size 12
- Subclass 28, size 9
- Subclass 29, size 14
- Subclass 30, size 18
- Subclass 31, size 5
- Subclass 32, size 20
- Subclass 33, size 3
- Subclass 34, size 4
- Subclass 35, size 1
- Subclass 36, size 3
- Subclass 37, size 3
- Subclass 38, size 3
- Subclass 39, size 4
- Subclass 40, size 2
- Subclass 41, size 2
- Subclass 42, size 2
- Subclass 43, size 2
- Subclass 44, size 13
- Subclass 45, size 10
- Subclass 46, size 8
- Subclass 47, size 1

- Subclass 48, size 11
- Subclass 49, size 1
- Subclass 50, size 3
- Subclass 51, size 11
- Subclass 52, size 5
- Subclass 53, size 5
- Subclass 54, size 7
- Subclass 55, size 5
- Subclass 56, size 6
- Subclass 57, size 8
- Subclass 58, size 10

Metaclass 1, size 18:

- Subclass 0, size 2
- Subclass 1, size 4
- Subclass 2, size 1
- Subclass 3, size 3
- Subclass 4, size 4
- Subclass 5, size 1
- Subclass 6, size 1
- Subclass 7, size 2
- Subclass 8, size 6
- Subclass 9, size 2
- Subclass 10, size 3
- Subclass 11, size 1
- Subclass 12, size 1
- Subclass 13, size 6
- Subclass 14, size 1
- Subclass 15, size 7
- Subclass 16, size 4

- Subclass 17, size 1

Metaclass 2, size 39:

- Subclass 0, size 15
- Subclass 1, size 19
- Subclass 2, size 12
- Subclass 3, size 11
- Subclass 4, size 17
- Subclass 5, size 14
- Subclass 6, size 17
- Subclass 7, size 37
- Subclass 8, size 37
- Subclass 9, size 14
- Subclass 10, size 24
- Subclass 11, size 13
- Subclass 12, size 22
- Subclass 13, size 49
- Subclass 14, size 29
- Subclass 15, size 35
- Subclass 16, size 5
- Subclass 17, size 10
- Subclass 18, size 7
- Subclass 19, size 10
- Subclass 20, size 13
- Subclass 21, size 20
- Subclass 22, size 18
- Subclass 23, size 6
- Subclass 24, size 2
- Subclass 25, size 6
- Subclass 26, size 3

- Subclass 27, size 1
- Subclass 28, size 3
- Subclass 29, size 4
- Subclass 30, size 12
- Subclass 31, size 4
- Subclass 32, size 17
- Subclass 33, size 11
- Subclass 34, size 31
- Subclass 35, size 8
- Subclass 36, size 4
- Subclass 37, size 15
- Subclass 38, size 1

Metaclass 3, size 7:

- Subclass 0, size 15
- Subclass 1, size 24
- Subclass 2, size 25
- Subclass 3, size 16
- Subclass 4, size 23
- Subclass 5, size 22
- Subclass 6, size 31

Metaclass 4, size 36:

- Subclass 0, size 7
- Subclass 1, size 6
- Subclass 2, size 4
- Subclass 3, size 11
- Subclass 4, size 4
- Subclass 5, size 8
- Subclass 6, size 6
- Subclass 7, size 6

- Subclass 8, size 2
- Subclass 9, size 7
- Subclass 10, size 6
- Subclass 11, size 8
- Subclass 12, size 7
- Subclass 13, size 5
- Subclass 14, size 5
- Subclass 15, size 3
- Subclass 16, size 3
- Subclass 17, size 2
- Subclass 18, size 2
- Subclass 19, size 2
- Subclass 20, size 4
- Subclass 21, size 4
- Subclass 22, size 6
- Subclass 23, size 4
- Subclass 24, size 2
- Subclass 25, size 2
- Subclass 26, size 1
- Subclass 27, size 1
- Subclass 28, size 7
- Subclass 29, size 4
- Subclass 30, size 1
- Subclass 31, size 2
- Subclass 32, size 1
- Subclass 33, size 2
- Subclass 34, size 6
- Subclass 35, size 1

Metaclass 5, size 36:

- Subclass 0, size 2
- Subclass 1, size 1
- Subclass 2, size 1
- Subclass 3, size 1
- Subclass 4, size 2
- Subclass 5, size 3
- Subclass 6, size 1
- Subclass 7, size 1
- Subclass 8, size 2
- Subclass 9, size 4
- Subclass 10, size 2
- Subclass 11, size 4
- Subclass 12, size 1
- Subclass 13, size 4
- Subclass 14, size 2
- Subclass 15, size 2
- Subclass 16, size 3
- Subclass 17, size 2
- Subclass 18, size 1
- Subclass 19, size 1
- Subclass 20, size 1
- Subclass 21, size 2
- Subclass 22, size 1
- Subclass 23, size 1
- Subclass 24, size 1
- Subclass 25, size 1
- Subclass 26, size 2
- Subclass 27, size 3
- Subclass 28, size 3

- Subclass 29, size 2
- Subclass 30, size 2
- Subclass 31, size 1
- Subclass 32, size 1
- Subclass 33, size 1
- Subclass 34, size 1
- Subclass 35, size 1

Metaclass 6, size 53:

- Subclass 0, size 3
- Subclass 1, size 3
- Subclass 2, size 3
- Subclass 3, size 2
- Subclass 4, size 3
- Subclass 5, size 2
- Subclass 6, size 3
- Subclass 7, size 1
- Subclass 8, size 1
- Subclass 9, size 1
- Subclass 10, size 1
- Subclass 11, size 1
- Subclass 12, size 1
- Subclass 13, size 2
- Subclass 14, size 2
- Subclass 15, size 1
- Subclass 16, size 2
- Subclass 17, size 5
- Subclass 18, size 2
- Subclass 19, size 2
- Subclass 20, size 4

- Subclass 21, size 2
- Subclass 22, size 3
- Subclass 23, size 2
- Subclass 24, size 1
- Subclass 25, size 3
- Subclass 26, size 2
- Subclass 27, size 3
- Subclass 28, size 4
- Subclass 29, size 1
- Subclass 30, size 1
- Subclass 31, size 1
- Subclass 32, size 2
- Subclass 33, size 1
- Subclass 34, size 2
- Subclass 35, size 2
- Subclass 36, size 2
- Subclass 37, size 2
- Subclass 38, size 2
- Subclass 39, size 1
- Subclass 40, size 2
- Subclass 41, size 3
- Subclass 42, size 2
- Subclass 43, size 3
- Subclass 44, size 1
- Subclass 45, size 2
- Subclass 46, size 2
- Subclass 47, size 2
- Subclass 48, size 3
- Subclass 49, size 3

- Subclass 50, size 3
- Subclass 51, size 2
- Subclass 52, size 5

Metaclass 7, size 26:

- Subclass 0, size 4
- Subclass 1, size 5
- Subclass 2, size 6
- Subclass 3, size 2
- Subclass 4, size 5
- Subclass 5, size 5
- Subclass 6, size 7
- Subclass 7, size 4
- Subclass 8, size 4
- Subclass 9, size 2
- Subclass 10, size 3
- Subclass 11, size 11
- Subclass 12, size 12
- Subclass 13, size 8
- Subclass 14, size 2
- Subclass 15, size 4
- Subclass 16, size 16
- Subclass 17, size 14
- Subclass 18, size 10
- Subclass 19, size 7
- Subclass 20, size 5
- Subclass 21, size 6
- Subclass 22, size 6
- Subclass 23, size 8
- Subclass 24, size 5

- Subclass 25, size 9

Metaclass 8, size 33:

- Subclass 0, size 3
- Subclass 1, size 4
- Subclass 2, size 5
- Subclass 3, size 5
- Subclass 4, size 2
- Subclass 5, size 4
- Subclass 6, size 3
- Subclass 7, size 5
- Subclass 8, size 3
- Subclass 9, size 5
- Subclass 10, size 3
- Subclass 11, size 7
- Subclass 12, size 5
- Subclass 13, size 9
- Subclass 14, size 3
- Subclass 15, size 5
- Subclass 16, size 5
- Subclass 17, size 5
- Subclass 18, size 1
- Subclass 19, size 3
- Subclass 20, size 4
- Subclass 21, size 3
- Subclass 22, size 5
- Subclass 23, size 9
- Subclass 24, size 5
- Subclass 25, size 9
- Subclass 26, size 8

- Subclass 27, size 2
- Subclass 28, size 5
- Subclass 29, size 6
- Subclass 30, size 4
- Subclass 31, size 2
- Subclass 32, size 3

Metaclass 9, size 33:

- Subclass 0, size 19
- Subclass 1, size 21
- Subclass 2, size 34
- Subclass 3, size 7
- Subclass 4, size 5
- Subclass 5, size 1
- Subclass 6, size 23
- Subclass 7, size 8
- Subclass 8, size 16
- Subclass 9, size 3
- Subclass 10, size 2
- Subclass 11, size 1
- Subclass 12, size 8
- Subclass 13, size 8
- Subclass 14, size 13
- Subclass 15, size 41
- Subclass 16, size 32
- Subclass 17, size 21
- Subclass 18, size 37
- Subclass 19, size 16
- Subclass 20, size 52
- Subclass 21, size 5

- Subclass 22, size 3
- Subclass 23, size 4
- Subclass 24, size 11
- Subclass 25, size 20
- Subclass 26, size 19
- Subclass 27, size 22
- Subclass 28, size 38
- Subclass 29, size 17
- Subclass 30, size 18
- Subclass 31, size 9
- Subclass 32, size 47

Metaclass 10, size 12:

- Subclass 0, size 8
- Subclass 1, size 31
- Subclass 2, size 40
- Subclass 3, size 20
- Subclass 4, size 10
- Subclass 5, size 26
- Subclass 6, size 10
- Subclass 7, size 21
- Subclass 8, size 19
- Subclass 9, size 13
- Subclass 10, size 10
- Subclass 11, size 28

Metaclass 11, size 44:

- Subclass 0, size 8
- Subclass 1, size 6
- Subclass 2, size 9
- Subclass 3, size 3

- Subclass 4, size 7
- Subclass 5, size 2
- Subclass 6, size 5
- Subclass 7, size 5
- Subclass 8, size 2
- Subclass 9, size 4
- Subclass 10, size 3
- Subclass 11, size 4
- Subclass 12, size 2
- Subclass 13, size 3
- Subclass 14, size 3
- Subclass 15, size 4
- Subclass 16, size 4
- Subclass 17, size 4
- Subclass 18, size 3
- Subclass 19, size 3
- Subclass 20, size 5
- Subclass 21, size 4
- Subclass 22, size 11
- Subclass 23, size 14
- Subclass 24, size 7
- Subclass 25, size 3
- Subclass 26, size 4
- Subclass 27, size 6
- Subclass 28, size 8
- Subclass 29, size 20
- Subclass 30, size 14
- Subclass 31, size 5
- Subclass 32, size 6

- Subclass 33, size 18
- Subclass 34, size 30
- Subclass 35, size 10
- Subclass 36, size 18
- Subclass 37, size 10
- Subclass 38, size 7
- Subclass 39, size 6
- Subclass 40, size 8
- Subclass 41, size 18
- Subclass 42, size 6
- Subclass 43, size 16

Metaclass 12, size 3:

- Subclass 0, size 1
- Subclass 1, size 1
- Subclass 2, size 1

Metaclass 13, size 1:

- Subclass 0, size 1

Metaclass 14, size 4:

- Subclass 0, size 1
- Subclass 1, size 1
- Subclass 2, size 1
- Subclass 3, size 1

Metaclass 15, size 50:

- Subclass 0, size 1
- Subclass 1, size 1
- Subclass 2, size 1
- Subclass 3, size 1
- Subclass 4, size 1
- Subclass 5, size 1

- Subclass 6, size 1
- Subclass 7, size 1
- Subclass 8, size 1
- Subclass 9, size 1
- Subclass 10, size 1
- Subclass 11, size 1
- Subclass 12, size 1
- Subclass 13, size 2
- Subclass 14, size 1
- Subclass 15, size 2
- Subclass 16, size 1
- Subclass 17, size 1
- Subclass 18, size 2
- Subclass 19, size 2
- Subclass 20, size 1
- Subclass 21, size 1
- Subclass 22, size 2
- Subclass 23, size 1
- Subclass 24, size 2
- Subclass 25, size 2
- Subclass 26, size 1
- Subclass 27, size 2
- Subclass 28, size 2
- Subclass 29, size 1
- Subclass 30, size 2
- Subclass 31, size 1
- Subclass 32, size 2
- Subclass 33, size 1
- Subclass 34, size 2

- Subclass 35, size 1
- Subclass 36, size 1
- Subclass 37, size 1
- Subclass 38, size 1
- Subclass 39, size 1
- Subclass 40, size 1
- Subclass 41, size 1
- Subclass 42, size 1
- Subclass 43, size 1
- Subclass 44, size 1
- Subclass 45, size 1
- Subclass 46, size 1
- Subclass 47, size 1
- Subclass 48, size 1
- Subclass 49, size 1

Metaclass 16, size 11:

- Subclass 0, size 2
- Subclass 1, size 1
- Subclass 2, size 1
- Subclass 3, size 2
- Subclass 4, size 1
- Subclass 5, size 1
- Subclass 6, size 1
- Subclass 7, size 1
- Subclass 8, size 1
- Subclass 9, size 1
- Subclass 10, size 1

Metaclass 17, size 43:

- Subclass 0, size 2

- Subclass 1, size 2
- Subclass 2, size 2
- Subclass 3, size 2
- Subclass 4, size 1
- Subclass 5, size 1
- Subclass 6, size 1
- Subclass 7, size 3
- Subclass 8, size 2
- Subclass 9, size 2
- Subclass 10, size 2
- Subclass 11, size 1
- Subclass 12, size 1
- Subclass 13, size 2
- Subclass 14, size 1
- Subclass 15, size 1
- Subclass 16, size 2
- Subclass 17, size 2
- Subclass 18, size 2
- Subclass 19, size 2
- Subclass 20, size 3
- Subclass 21, size 1
- Subclass 22, size 2
- Subclass 23, size 1
- Subclass 24, size 1
- Subclass 25, size 2
- Subclass 26, size 1
- Subclass 27, size 1
- Subclass 28, size 2
- Subclass 29, size 2

- Subclass 30, size 2
- Subclass 31, size 4
- Subclass 32, size 2
- Subclass 33, size 1
- Subclass 34, size 5
- Subclass 35, size 3
- Subclass 36, size 2
- Subclass 37, size 1
- Subclass 38, size 3
- Subclass 39, size 3
- Subclass 40, size 3
- Subclass 41, size 2
- Subclass 42, size 2

Metaclass 18, size 43:

- Subclass 0, size 2
- Subclass 1, size 2
- Subclass 2, size 3
- Subclass 3, size 3
- Subclass 4, size 3
- Subclass 5, size 4
- Subclass 6, size 1
- Subclass 7, size 1
- Subclass 8, size 2
- Subclass 9, size 2
- Subclass 10, size 1
- Subclass 11, size 3
- Subclass 12, size 4
- Subclass 13, size 3
- Subclass 14, size 2

- Subclass 15, size 2
- Subclass 16, size 1
- Subclass 17, size 1
- Subclass 18, size 1
- Subclass 19, size 3
- Subclass 20, size 1
- Subclass 21, size 2
- Subclass 22, size 3
- Subclass 23, size 2
- Subclass 24, size 2
- Subclass 25, size 3
- Subclass 26, size 5
- Subclass 27, size 2
- Subclass 28, size 2
- Subclass 29, size 2
- Subclass 30, size 2
- Subclass 31, size 4
- Subclass 32, size 2
- Subclass 33, size 2
- Subclass 34, size 3
- Subclass 35, size 2
- Subclass 36, size 4
- Subclass 37, size 4
- Subclass 38, size 3
- Subclass 39, size 5
- Subclass 40, size 9
- Subclass 41, size 2
- Subclass 42, size 4

Metaclass 19, size 52:

- Subclass 0, size 1
- Subclass 1, size 1
- Subclass 2, size 2
- Subclass 3, size 1
- Subclass 4, size 2
- Subclass 5, size 1
- Subclass 6, size 3
- Subclass 7, size 1
- Subclass 8, size 1
- Subclass 9, size 1
- Subclass 10, size 1
- Subclass 11, size 1
- Subclass 12, size 1
- Subclass 13, size 1
- Subclass 14, size 1
- Subclass 15, size 1
- Subclass 16, size 1
- Subclass 17, size 1
- Subclass 18, size 1
- Subclass 19, size 1
- Subclass 20, size 1
- Subclass 21, size 2
- Subclass 22, size 1
- Subclass 23, size 2
- Subclass 24, size 2
- Subclass 25, size 2
- Subclass 26, size 2
- Subclass 27, size 1
- Subclass 28, size 1

- Subclass 29, size 2
- Subclass 30, size 2
- Subclass 31, size 1
- Subclass 32, size 2
- Subclass 33, size 1
- Subclass 34, size 2
- Subclass 35, size 1
- Subclass 36, size 2
- Subclass 37, size 1
- Subclass 38, size 1
- Subclass 39, size 2
- Subclass 40, size 1
- Subclass 41, size 1
- Subclass 42, size 3
- Subclass 43, size 1
- Subclass 44, size 1
- Subclass 45, size 1
- Subclass 46, size 1
- Subclass 47, size 2
- Subclass 48, size 2
- Subclass 49, size 2
- Subclass 50, size 3
- Subclass 51, size 1