Nuclear Engineering and Technology xxx (xxxx) xxx

Contents lists available at ScienceDirect

Nuclear Engineering and Technology

journal homepage: www.elsevier.com/locate/net

Original Article

Machine-learning methods for blind characterisation of nuclear fuel assemblies

J. Paz-Peñuelas-Oliván [a,*], J. Ruz [b,c,d,*]

[a] Facultad de Ciencias, Universidad de Zaragoza, 50009, Spain
[b] Fakultät für Physik, Technische Universität Dortmund, Dortmund, D-44221, Germany
[c] Also, Centro de Astropartículas y Física de Altas Energías (CAPA) & Departamento de Física Teórica, Universidad de Zaragoza, 50009, Spain
[d] Also, Physics and Life Science Directorate, Lawrence Livermore National Laboratory, Livermore 94550, CA, United States of America

ARTICLE INFO

Keywords:
Nuclear assemblies
Non-destructive analysis
Mathematical methods
Machine learning
Neural networks

ABSTRACT

The global prevalence of uranium as fissile fuel in nuclear reactors, paired with its transmutation to plutonium inside the core in an isotopic ratio corresponding to a *direct-use material* in *significant quantities*, makes the handling of spent nuclear fuel an important and sensitive matter towards non-proliferation efforts. A fast and reliable method for characterising spent fuel is thus desirable for spent nuclear fuel reprocessing and storage facilities.

We propose a non-destructive, blind and fast measure method of the quantity of spent nuclear fuel inside an assembly. By measuring photon fluency collectively for the complete assembly we determine the number of present fuel rods without the need to open the array and manually check. For this, we circle a detector set-up around the assembly and feed its measurements into a neural network for a prediction. Different specifically designed architectures based on dense and convolutional layers are trained on synthetically generated data using self-developed code on *python*. We arrive at the election of a convolutional network for optimal results.

We achieve an exact prediction with over three sigmas of confidence (99.85% accuracy) thanks to the double detector set-up we introduce in this article, proving the prediction power of neural networks in this instance with a relatively simple measure configuration.

1. Introduction

Currently, all nuclear power plants in use take advantage of fission, the process of radioactive decay through which a heavy nucleus splits into two or more smaller daughter nuclei. This process can take place spontaneously or it can be induced by shooting a neutron at the nucleus. One such fissile nuclei is the uranium isotope ^{235}U, which upon collision with a neutron releases high amounts of energy and an extra trio of neutrons that will make the sustained chain reaction possible.

The natural abundance of ^{235}U is rather low, with only a 0.71% against the 99.28% of the heavier isotope ^{238}U, which does not undergo fission as readily; so it is common that the uranium has to be enriched to power a reactor. This enrichment can vary from 2 to under 5%, although some reactors such as the CANDU (CANadian Deuterium Uranium) reactors run on natural uranium thanks to their use of heavy water as moderator. Once the uranium has been correspondingly enriched it is manufactured into *pellets*: little cylinders 1 cm high and about 0.8 cm in diameter, which are then stacked on top of each other inside a several-meter-long rod, called a *pin* or *rod*. These pins are mounted into arrays of different shapes according to design, making a fuel *assembly*, and a number of these assemblies make up the reactor core [1]. The shape and size of fuel assemblies vary with design, but the most widespread reactor type around the globe is the Pressurised Water Reactor (PWR), thus their assemblies are the most common. These are square assemblies of 17 pins per side, totalling 289 rod slots, of which 24 are employed for control rods, rather than fuel, which are necessary to manage the fission rates inside the reactor (see Fig. 1).

The uranium will spend a few years in the reactor core, usually around three, before being replaced by a fresh batch of material. The spent nuclear fuel is then temporarily placed in the reactor pools and later transported to the corresponding radioactive material storage

* Corresponding authors.
E-mail addresses: jorgepazp@gmail.com (J. Paz-Peñuelas-Oliván), Jaime.Ruz@cern.ch (J. Ruz).

https://doi.org/10.1016/j.net.2025.103462
Received 25 July 2024; Received in revised form 30 November 2024; Accepted 6 January 2025
Available online 13 January 2025

Acronyms

mse	mean squared error
CNN	Convolutional Neural Networks
DNN	Dense Neural Networks
ESS	Energy-Specific Study
IAEA	International Atomic Energy Association
IFEL	Irradiated Fuel Examination Laboratory
ORNL	Oak Ridge National Laboratory
PWR	Pressurised Water Reactor
SGD	Stochastic Gradient Descent
XRF	nuclear fluorescence

facility. However, this is not without challenges. In addition to being radioactive, spent nuclear uranium fuel contains plutonium formed by transmutation of ^{238}U upon impact with a neutron.[1] According to the International Atomic Energy Association (IAEA), plutonium containing less than 80% ^{238}Pu is a *direct use material*, that is "nuclear material that can be used for the manufacture of nuclear explosive devices without transmutation or further enrichment" [3], which is the case for spent nuclear fuel [4]. Additionally, it also contains more than a *significant quantity* of it: "the approximate amount of nuclear material for which the possibility of manufacturing a nuclear explosive device cannot be excluded", which for plutonium is 8 kilograms [3], as despite its 1% content in plutonium, a reactor core often contains several tonnes of fuel.

It is clear then, that a correct, responsible, safe and controlled handling of spent nuclear fuel is an important security and safeguarding concern, especially during transport from the reactor to the storage or reprocessing facility. Currently, plutonium content in spent fuel discharged from a reactor is only estimated through simulations and measured at arrival at the facilities through destructive methods (that is, the uranium rods are chemically broken down) [2]. This methodology has been observed to present shipper-receiver differences, most likely due to inaccuracies in the simulation, but an opportunity to take advantage of them arises. A fast and non-destructive method of measuring the presence of plutonium in spent fuel to verify that none of it is missing is an important milestone in non-proliferation efforts. This directly implies a need to rapidly and non-destructively measure quantities of spent nuclear fuel content.

Machine learning methods, and in particular Neural Networks, have been successfully implemented in recent publications and advances on the spent nuclear fuel characterisation and safeguarding fields [5–11], as well as other related nuclear research [12–16]. We propose a new measurement procedure that will rely on neural networks to blindly determine the number of present burnt uranium pins inside an assembly. Under our knowledge, this has not been done before.

1.1. Non-destructive nuclear pin measurements

As mentioned above, we will work with a non-destructive approach to measure the content of a spent nuclear assembly, in particular one that takes advantage of the gamma decay and the X-rays emitted thanks to nuclear fluorescence (XRF), a phenomenon that occurs in spent nuclear fuel due to the more energetic gamma decay of fission products (see Fig. 2). This has the additional advantage that it can be performed without having direct access to the spent fuel, allowing us to measure an enclosed assembly.

[1] The ^{238}U transmutes to ^{239}U, which beta decays to ^{239}Np. This nucleus beta decays again to ^{239}Pu. Other plutonium isotopes are formed through similar decay chains, but ^{239}Pu retains the highest final abundance [2].

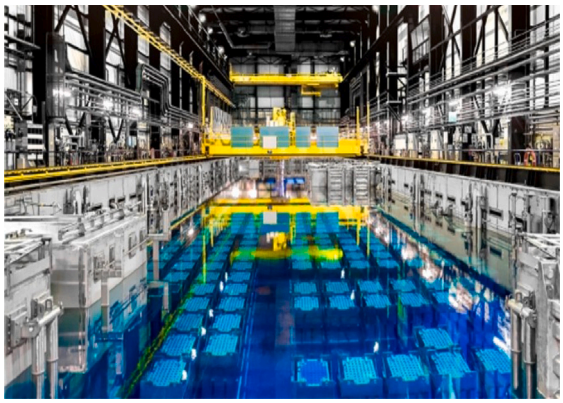

Fig. 1. Reactor pools.

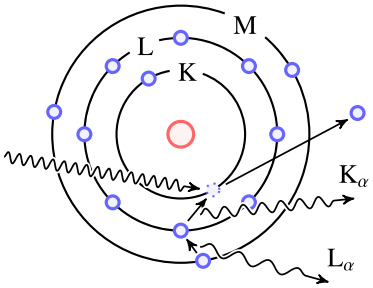

Fig. 2. Simple representation of the XRF cascade following an electron emission in aluminium.

Table 1
Characteristic XRF emissions for uranium and plutonium: energy and ratio.
Source: Data from Ref. [17].

XRF	Uranium		Plutonium	
	E (keV)	ratio (%)	*E* (keV)	ratio (%)
$K_{\alpha 1}$	98.44	100.00	103.76	100.00
$K_{\alpha 2}$	94.67	61.90	99.55	62.50
$K_{\beta 1}$	111.30	22.00	117.26	22.20
$K_{\beta 2}$	114.50	12.30	120.60	12.50
$K_{\beta 3}$	110.41	11.60	116.27	11.70

The energies of the emitted X-rays are dependent only on atomic number, making them characteristic of each element and blind to isotopic composition. XRF analysis is a widely used method for element composition identification in materials that takes advantage of this phenomenon. In the case of uranium and plutonium, all K-emissions fall within the 94 to 121 keV range, and they all have a very specific branching-ratio (see Table 1 for a summary of Uranium and Plutonium fluorescence).

Historically, XRF for elemental composition analysis was difficult to carry out on spent nuclear fuel due to the small percentage of plutonium in the rods, in addition to the high amounts of other fission products making an important contribution to the background. Furthermore, the $K_{\alpha 1}$ line of Plutonium (its most intense one) lays very close energetically to the gamma decay energy of ^{155}Eu, also present in the fuel rod in similar quantities to plutonium (the energies are 103.8 and 105.3 keV respectively), so detectors with considerably good energy resolution are necessary to tell the two emissions apart.

This measurement was achieved at the Oak Ridge National Laboratory (ORNL) Irradiated Fuel Examination Laboratory (IFEL) through the use of gamma-ray mirrors [18–20]. These mirrors acted as band-pass filters in the range of energy of the K X-ray emissions, allowing

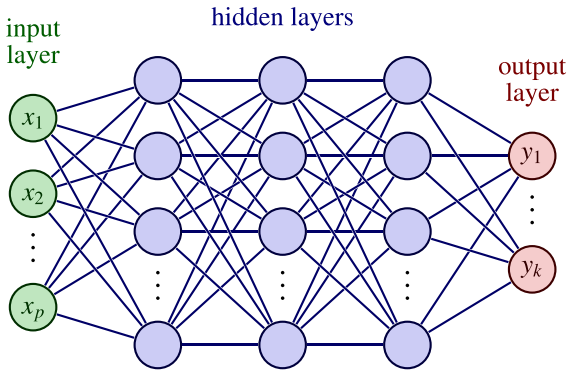

Fig. 3. Schematic of a dense neural network. Data travels from left to right. Modified from original by Izaak Neutelings.

for an increase in resolution and a measurement of the Pu-U ratio with very little uncertainty (less than 0.1%) [20]. It is these measurements that inspired the method presented in this article.

1.2. Neural networks

A neural network is an algorithm that mimics how the human brain works. They were envisioned as a different approach to problems that classical programming techniques were not able to resolve [21]. Let us begin by establishing the mathematical model of a neuron, which is a generalisation of the original *perceptron* introduced by Frank Rosenblatt [22]. A mathematical neuron is a function

$$\begin{array}{ccc} \mathbb{R}^p & \longrightarrow & \mathbb{R} \\ \mathbf{x} = \{x_p\} & \mapsto & f(\mathbf{w}^T\mathbf{x} + b) \end{array} , \tag{1}$$

where $\mathbf{w} \in \mathbb{R}^p$ denotes the *weights* vector, b is the neuron *bias* and $f : \mathbb{R} \to \mathbb{R}$ is called the *activation function*. f can be various functions, including the identity, but will always be monotonically increasing and generally non-linear. The Heaviside step function, the sigmoid, or the hyperbolic tangent are common examples. The choice of activation function will determine the behaviour and success of a neuron, and thus the neural network, in a satisfactory fit. Apart from some general guidelines, the determination of the optimal activation function is heuristic and falls upon the neural network programmer.

1.2.1. Dense Neural Networks (DNN)

To put the mathematical neurons to use we can employ dense neural networks, or DNNs for short. These *feedforward* models arrange collections of neurons in *layers*. All neurons in a layer will take the same inputs, and all their outputs will be collectively fed into the next neuron layer, whose output will be fed into the next one, and so on until we arrive at the final output layer. Such a neural network model is represented in Fig. 3.

Note that the output can be a k-dimensional vector, and not necessarily a scalar. The layers between the input and output are called hidden layers, and there can be as many as the network architect desires, each with a different number of neurons. Usually, the activation function is common for all neurons in a layer. When a neural network has more than one or two hidden layers, we call it a deep neural network, and we say that we are doing deep learning.

It has been proven that a sufficiently deep and wide dense network can be used to approximate as closely as desired any function from $\mathbb{R}^p$ to $\mathbb{R}^k$ [23]. This is known as the Theorem of Universality, and it justifies the widespread interest in neural networks across increasingly numerous fields.

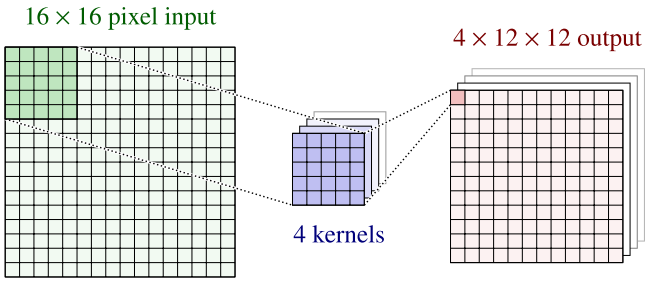

Fig. 4. Schematic of a convolutional layer with 5 by 5 kernels applied to a 16 by 16 pixel image.

1.2.2. Convolutional Neural Networks (CNN)

We have seen that in a dense neural network all neurons in a layer feed from the outputs of all neurons in the previous layer. This can sound very mathematically potent, but in practice, this structure may come short and inefficient when approaching certain problems, especially when it comes to image recognition.

Convolutional neural networks, or for short CNN, are network structures which take advantage of the spatial arrangement of the input data. When providing an image to a network we are giving it a matrix of values, hence it is sensible to consider the adjacency of pixels: after all, an image makes sense not by the individual pixel values but by the relative values in close spatial proximity. Thus, this structure is of particular advantage when adjacent measurements share a certain level of correlation. Let us explain how they work using Fig. 4.

Rather than having a collection of neurons with their individual weights, biases and outputs, a convolutional layer defines a set of filters, or *kernels*, which will swipe the input grid for an output matrix of a reduced dimension.

Fig. 4 shows a collection of 5 by 5 kernels, or filters, placed on the first submatrix of the input matrix, highlighted in green, to a single scalar output in red. This operation is computed as the sum of the components of the Hadamard product [24] of the input submatrix and the kernel, plus a kernel bias. Each element of the kernel can be understood as a positional weight, and they, together with the bias, remain constant when the kernel slides across the input matrix. Their values only change as the network learns. This has very interesting interpretations, mainly that each filter learns to see a feature anywhere on the image, but we will not get into that here.

A convolutional layer is a collection of filters which can contain as few as three or as many as a few tens or a hundred. Kernels can vary in size but are usually square and rarely bigger than 5 by 5, which means that when analysing high-definition images the size of the output can build up considerably. This build-up is avoided with the use of *pooling layers*, which reduce the dimension of the output by, simply put, reducing the resolution. A common choice is the *maxpooling* layer. Once the dimension has been reasonably reduced, usually after a few concatenations of convolutional and pooling layers, it is customary to flatten the data into a vector and feed it into a dense network for final fitting. For a more in-depth explanation of convolutional layers and networks the reader is referred to [21].

1.3. Neural network learning

To train a neural network we provide it with a set of training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$. With it, we aim to minimise the cost, or loss, function, generally defined as the mean squared error (*mse*), by adjusting the values of the weights and bias of each neuron.

$$\text{mse}(w, b) = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{a}_i\|^2 , \tag{2}$$

where $\mathbf{a}_i$ represents the neural network output, and depends on $\mathbf{x}_i, w$ and b. Here we have denoted by w and b the collection of all weights

	Predicted Positives (PP)	Predicted Negatives (PN)
Positives (P)	True Positives (TP)	False Negative (FN)
Negatives (N)	False Positives (FP)	True Negatives (TN)

Fig. 5. Confusion matrix C for the binary classification case (positive vs negative).

and biases respectively of all neurons in the network. Recall that $\mathbf{x}_i$ and $\mathbf{y}_i$ are fixed data, not trainable parameters, thus the lack of dependency of the *mse* with either. By minimising the cost function, as $\mathbf{a}_i$ tends to $\mathbf{y}_i$, the network is said to learn. The *mse* is a very frequent choice in *regression* problems, but other losses can be defined and used as target function. In *classification* problems, where rather than a numerical prediction we are attempting to discern within discrete categories, the most common loss function is the *cross-entropy* [21]. In these cases, it is important that the training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ has a comparable number of instances in each category to avoid the network developing a bias towards the over-represented class when minimising the cost function.

The way to go about minimising the loss function is also a choice for the network programmer. There exist plenty of optimisation algorithms, but the most widespread are the Stochastic Gradient Descent (SGD) and the Adam algorithm [25].

1.4. Performance metrics

Once a network is trained, the simplest way of determining performance is the value of the cost function: the lower the better. The mse, shown in Eq. (2), is quite useful in regression problems where we want to fit a continuous function, but in classification problems as is our case as we just discussed, it is not very intuitive due to the lack of information on the success of the classification.

In these cases, it is common practice to work with the confusion matrix C, which is a matrix where the component C_{ij} (ith row and jth column) corresponds to the number of entries whose true class is i and are predicted to belong to class j. Thus, in a perfect classification C would be a diagonal matrix. Historically the confusion matrix was introduced for binary problems, and thus it is usually represented as in Fig. 5.

From this representation one defines the *Precision* and the *Recall* as

$$\text{Precision} = \frac{TP}{PP} = \frac{TP}{TP+FP}\,, \quad \text{Recall} = \frac{TP}{P} = \frac{TP}{TP+FN}\,.$$

Thus, precision indicates the probability of a positive prediction actually being positive, and the recall the probability of a positive being identified as such [26]. In problems where a perfect classification is not attained, one may choose to modify the threshold to prioritise the maximisation of one of these metrics over the other. These two metrics can be extended to the multiclass classification problem by defining positive the considered class and negative all the others, thus obtaining class-dependent metrics. These can be later averaged out for an overall outlook.

1.5. Working with neural networks

The first step of implementing a neural network is its design and definition, that is, the declaration of the different layers and their corresponding activation functions, and the choice of the cost function together with the optimiser. This process involves the establishment of a couple of *hyperparameters*, which are constants that will influence the numerical performance of the network. Such hyperparameters include the *learning rate* η, which in general terms establishes how far down the gradient we will move in each optimisation step, the *batch-size*, which is the number of data entries considered when computing the gradient (this is done for computational speed and memory efficiency), and the number of *epochs*, which is the number of optimisation runs through the complete dataset the networks performs before being considered trained. The determination of their value is purely heuristic again.

Once the neural network has been declared it is time to feed it the dataset. However, it is always good practice to normalise and shuffle the available data and split it into *training set* and *validation set*. This separation is done for network performance checks, mainly to verify that the network has not over fitted (that is, learnt the noise in the data), which can become a complication due to the high number of parameters. If the neural network is as successful with the training set as with the validation set, within reasonable statistics, then we can discard over fitting.

2. Fuel assembly characterisation

In the conclusions of Ref. [20] it is mentioned that a 0.1% precision measurement of a fuel pin can be made in a 20-min interval. When taking into account the number of rods in a standard PWR assembly (265) and the number of assemblies that go into a reactor each time it is loaded (typically between 120 and 200), in addition to all the necessary fuel handling, which not only takes time but also needs skilled operators, single pin measurements could last incredibly long. For reference, the time to analyse all waste produced in a reactor appears to be comparable to the time it takes to burn the load in the first place. This, in addition to the already existing spent fuel assemblies of the last 50 years of reactor operation, would represent an impractically long time to have all nuclear residue accounted for.

2.1. Objectives

We propose a method that allows to greatly speed up these measurements. The idea is to circle a detector around a whole assembly, which is encased such that the total amount of pins present and their position is unknown, making a fixed number of measurements at different angular positions. These are then fed to a neural network to attempt to deduce information about the hidden assembly.

Our goal is the development of a mathematical tool to blindly predict the number of pins present in an assembly given a set of equally-spaced photon fluence measurements. Thus, it represents a very interesting proposition: a simple measurement of a concealed spent fuel assembly to determine with very little uncertainty the exact number of present pins. This poses to be extremely helpful in the shipper-receiver difference mentioned earlier for nuclear residue accountancy, with a measurement taken when leaving the reactor pools and again at the storage or reprocessing facilities. This approach is possible as these photon emission measurements can be taken without opening the assembly enclosure. It can later be compounded with a few single-pin measurements for a more precise estimation of the content of plutonium on a reactor offload batch, providing us with a wider toolkit to choose where we wish to fall on the balance between measurement precision and time spent.

In addition to developing the neural network, we have written a library to simulate the assembly and the spectra read at the detector, with which we have numerically generated synthetic data to train our network. This has been an elemental approach to keep the amount of work manageable and within scope. We have not considered the different sources of photon attenuation that would affect the measured spectre at the detector, and we have instead focused on the spectra's collective contribution, among other simplifying approximations. In the conclusions, we briefly discuss the different approaches to the problem if attenuation were to be taken into account and the difficulties and challenges it would pose for the measurement.

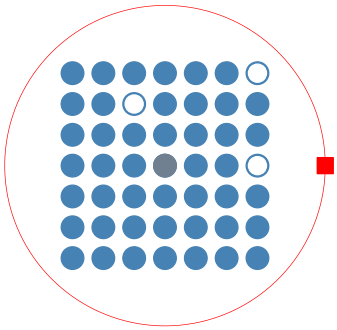

Fig. 6. A detector circling a smaller 7 by 7 fuel assembly missing 3 pins. In practice, we would not be able to see the assembly.

2.2. Mathematical description

We consider PWR square assemblies with s pins per side, s being an odd number, totalling s^2 slots in the assembly. For simplicity, we assume that no pin slot is permanently kept empty for control rods, that is, that all s^2 slots can contain a fuel pin; however, including control rods should be easy using the developed code. Another simplifying assumption made is considering only odd values of s, but it is worth mentioning that assemblies with an even number of pins per side also exist, although they are not as common. We will also discuss smaller assemblies than 17 by 17, namely as the work was initially done with a 7 by 7 assembly to keep execution times manageable while the code was being written (s^2 for $s = 7$ is 49, against the 289 slots of a 17 by 17 assembly). Results will be given for $s \in \{3, 5, 7, \dots, 17\}$, with special attention to the last, and we will see that they depend heavily on the value of s. Furthermore, when talking about non-complete assemblies, we denote the number of missing pins by m, or the number of present pins with n, such that $n + m = s^2$.

Spectra and measurements. Each pin k, with $k \in \{1, \dots, s^2\}$, present in the assembly is assigned a known X-ray emission fluence spectrum ϕ_k. We have worked with three spectra measured at the ORNL IFEL [19,20] in the energy range of 1 to around 300 keV; to later assign convex linear combinations of them to model different spectra for each pin.

The measurements we have simulated are of a detector circling the assembly in the plane perpendicular to the fuel rods as seen in Fig. 6. Thus, the spectre Φ measured at the detector can be easily computed as

$$\Phi = \sum_{k=1}^{s^2} \left(\frac{D_k}{d_k} \right)^2 \phi_k \,, \tag{3}$$

where D_k is the distance at which the spectra had been originally measured [20] and d_k is the current distance from the pin to the detector. Note that Φ depends on the detector placement, which is given in polar coordinates (r, θ) with respect to the central pin of the assembly, but as our measurements keep the radius constant we can write $\Phi = \Phi(\theta)$. When talking about the fluence for a specific energy E we will write Φ_E, also dependent on θ. It is easy to convince oneself that Φ, and consequently Φ_E, are both continuous and 2π-periodic with respect to θ.

A note on combinatorics. Assume that we have an $s \times s$ assembly where all pins have been assigned the same spectrum ϕ. In the situation we wish to model, each pin can be present or missing, giving us 2^{s^2} total possible assemblies. Even at $s = 3$ this number rises to 512, but given the fact that a neural network can have up to a couple thousand parameters (counting all weights and biases), a correct classification of size 3 assemblies should not present any problems. However, when we increase the assembly size to $s = 17$ the narrative is completely different. At this size, the amount of possible assemblies rises to over 10^{86}, more than the cube of Avogadro's number.

Despite this, our goal is not to precisely classify the assemblies into the different possible arrangements, which would let us know where a pin is missing, but to predict the number of present pins. Now, say that there are $0 \le n \le s^2$ pins in the assembly. Then the number of possible arrangements is equal to the number of ways to place the n pins into the s^2 slots, or in other words the number of ways to choose n slots to be occupied out of the possible s^2. That is

$$\binom{s^2}{n} \,. \tag{4}$$

Notice that the above number is equal to $\binom{s^2}{m}$ as $n + m = s^2$, as the binomial coefficient is a symmetric function of n with respect to its maximum at $n = s^2/2$. Note that in our case, as s^2 is odd, this happens at $\lfloor s^2/2 \rfloor$ and at $\lceil s^2/2 \rceil$. Also, we have that $\binom{s^2}{0} = \binom{s^2}{s^2} = 1$, as one would naturally expect. Knowing this is quite helpful in regards to data generation for neural network training, as we discuss in Section 3.4.

It is also important to mention that if we take into account the rotational symmetry of the detector and how we can consider assemblies equal under 90° rotations, all the above quantities will have to be corrected by a 1/4 factor, but the set of possible assemblies remains incredibly large. Of course, this is amplified further when one considers different spectra for each of the pins, which is a more realistic hypothesis.

3. Computational framework

Our work aims to numerically simulate spent fuel assembly photon emissions to infer information about the assembly through machine learning algorithms. To do so we chose to make use of *python* [27], as it is the language where most machine learning suites are developed and readily available, namely the *tensorflow* and *keras* libraries [28], but also the *scikit-learn* library [29] for the metrics. Rather than pure code, for convenience, we have written a collection of *jupyter notebooks* [30], which we will mention as we go along.

Having said this, it is worth spending some time discussing the most basic premises of the self-developed *Nuclear Assemblies* library [31], as we have extensively used it for data generation. For replicability and interested readers, it is available at Github [32] with the pertinent explanations in markdown and commentary.

3.1. Nuclear assemblies python library

This library is the cornerstone of our work. In it, we define three python classes: the *spectre* class, the *assembly* class and the *detector* class, with their corresponding attributes, manipulation methods and computation and plotting algorithms.

The spectre class. This class contains three attributes: the name, the distance at which it was measured D_k, and the data itself. The `spectre` is a two-dimensional array containing the energies vector in MeV, the fluences vector ϕ_k in $\gamma / \mathrm{s} \cdot \mathrm{cm}^2 \cdot 0.5$ keV (photons per second per cm^2 per 0.5 keV), and the relative errors vector. Among the class methods, the only one worth mentioning here is the `spectre.draw()` method, which plots the spectrum.

The assembly class. The assembly class has four attributes: the size s, the physical distance between centres of adjacent pins (which defaults to 1.25 cm), the *pinslots* array, which contains data on whether the pin is occupied or not, and the *spectra* array, of the same dimension as pinslots to contain each pin's `spectre`'s name. The methods of this class are its most important feature, as they allow us to remove and add pins, manually or randomly, as well as other plotting methods which will be built upon by those of the detector class.

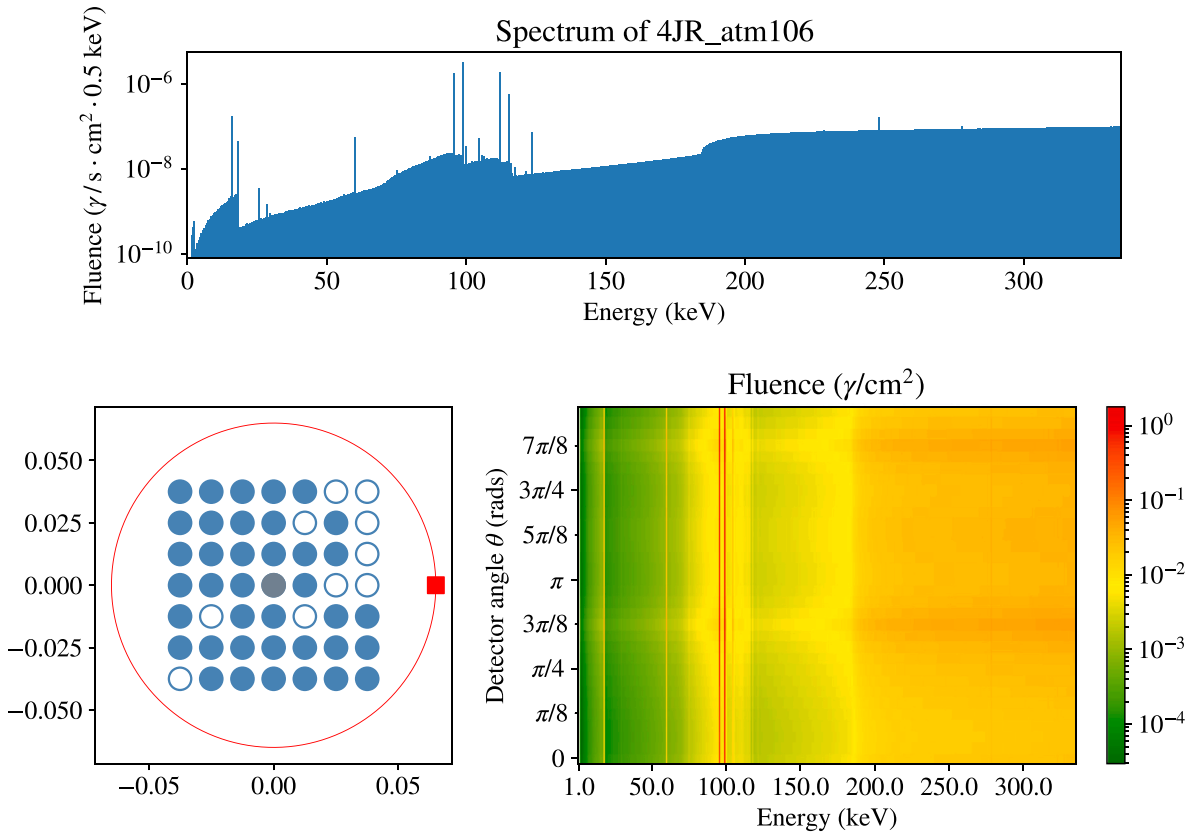

Fig. 7. Top: Plot of one of the provided nuclear pin XRF spectrum [19] using the `.draw()` method. **Bottom:** The full representation of a detector instance with the data entry it generates using the `.draw()` method. In this assembly, all the pins have been assigned the above `spectre`. Note that the `dataentry` plot is not completely accurate as it may have lost detail due to its size.

The detector class. The detector class is the most elaborate in the library. It has three main attributes: the assembly it circles, the detector's placement in the usual polar coordinates (r, θ), and the spectre $\Phi(r, \theta)$ it reads from the collective emissions of all the pins in the assembly at its current placement. A graphic representation of a detector class instance can be seen in Fig. 6.

In addition to inheriting all the assembly modification methods from the assembly class, this class also has methods that allow the user to interact with the detector's placement (modify it, obtain Cartesian coordinates, etc.) and of course to compute Φ through Eq. (3). Most interesting of all is the `detector.gen_dataentry()` method, which circles the detector around the assembly and generates a two-dimensional array that contains the detector's spectre at p equally-spaced angular positions, that is, we are computing $\Phi(\theta)$ for θ in $\{0, {}^{2\pi}/_{p}, {}^{4\pi}/_{p}, \dots, {}^{2(p-1)\pi}/_{p}\}$. The default is $p = 32$.

We invite the reader to look at Fig. 7 for a detailed example of `spectre`, `assembly` and `detector` classes.

3.2. General design of our neural network

In essence, we have implemented a classification network that has been optimised as a regression one. We need to consider it a classification problem since we are interested in predicting the exact number of pins $n \in \{1, \dots, s^2\}$, which is a natural number, and also because we will need to train the network with a similar number of assemblies for all values of n to prevent it from developing a bias towards a prediction. Nonetheless, we optimise it as a regression problem, as when given an assembly with $n = 26$ pins a prediction of 25 is far better than a prediction of 45, whereas if trained like a classification one (such as dogs vs cats), both predictions would be weighted as being equally wrong.

3.3. Defining performance metrics

In addition to the precision and the recall metrics defined in Section 1.4, which we will use for class-specific (n-dependant) performance, we will be defining a new metric for overall performance. In our case simply counting how many assemblies have had the number of present pins n correctly predicted is the most representative, which corresponds to the usual *accuracy* metric, but to characterise how close the failed predictions are to the real value, our function has the option to consider a prediction correct if it falls within an error of $e \in \mathbb{N}$ of the real value. We have called this metric `npins_accuracy`. Note that as the prediction will be continuous as per the above section, for metric computation we will simply undo the normalisation and round the obtained values to provide the final discrete prediction.

3.4. Generation of data

Recall that we are interested in being able to predict the number of pins given a detector rotation, that is, given the data entry on the bottom right of Fig. 7 predict the number of blue pins on its left. If we dedicate some thought to it, the information we are after lies in the coefficients of ϕ_k in Eq. (3), rather than in ϕ_k themselves. This means that we are not interested in the whole image that the `dataentry` computes, but instead only in the fluence for a specific energy E and its dependency with θ, which we agreed to call $\Phi_E(\theta)$. Therefore, each data entry we synthetically generate contains the $\Phi_E(\theta)$ tuple for $p = 32$ equally-spaced values of θ (making it a 32-component vector) and the corresponding pin slots array that generated it. We have called them our X and Y variables respectively,

As we discussed in , for an s-sized PWR assembly and for any possible value of n there exist $\binom{s^2}{n}$ different arrangements of assemblies,

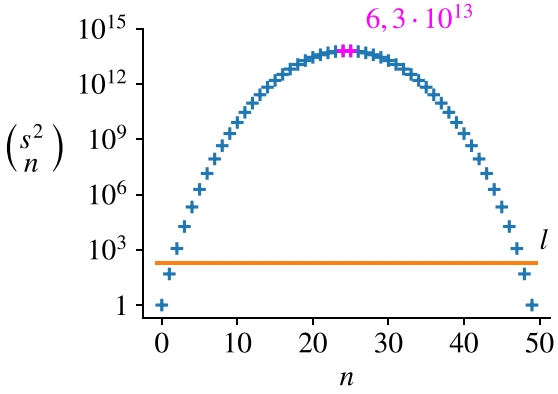

Fig. 8. Combinatorics and value of $l = 200$ for $s = 7$.

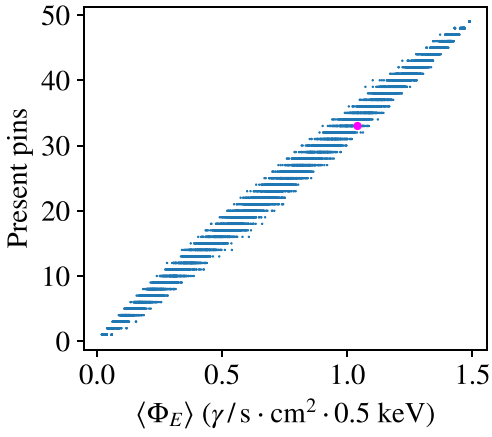

Fig. 9. Plot of the fluence *leaf*.

with maximum at $\lfloor s^2/2 \rfloor = {}^{(s^2-1)}/_2$. This means that we would need more data for each n the closer we are to the maximum for a proportionally equal representation of the probability of appearance of a certain assembly arrangement. However, we have explained that this value l must be fixed with respect to n to avoid bias in the network. Then it is a matter of selecting a big enough l to allow it to learn the patterns but small enough to make the data generation feasible. We heuristically fix $l = 200$ for all values of n (see Fig. 8), making the dataset corresponding to an s-size assembly contain ls^2 data entries. Note that data for $n = 0$ was not generated as trivially $\boldsymbol{\Phi}_E(\theta)\big|_{n=0}$ is the 0 vector.

The process of data generation was fairly simple. We began defining a detector instance with a complete assembly, which we call *baseDet*. In regards to the value of the radius of the detector circumference, we want it as close as physically possible to the assembly to maximise the variance of $\boldsymbol{\Phi}_E$ with respect to θ. Thus, we only left a small fixed margin from the corner pin to the detector to allow it to physically rotate as seen in Fig. 6. Once *baseDet* is defined, we iteratively copy it and randomly remove m pins l times for each value of $m = 0, \dots, s^2 - 1$. It is in this step where we could have decided to include the control rods: simply defining *baseDet* to not have fuel rods in the corresponding positions and iterate m from 0 to $s^2 - 1 - c$, where c is the number of control rods.

For each value of s to be studied ($s \in \{3, 5, \dots, 17\}$) we generated 2 datasets: One assigning the same spectrum to every pin, and one with convex linear combinations of the three available spectra (see Section 7), totalling 16 datasets. We want to emphasise that the data generation was done from the randomly generated assemblies, but this information is not accessible to the neural network, which is blind to the assembly structure. The options considered and final choice of the architecture of our neural network are explained in Section 4.2.

Finally, as we worked with $\boldsymbol{\Phi}_E(\theta)$, we decided to name this approach Energy-Specific Study (ESS), and the data generation has been realised through the *DataGen_ESS* notebook [31].

4. First results: 7 by 7 assemblies

We will begin with discussing the 7 by 7 assemblies due to the more evident phenomenology they present. The work in this section can be followed in the *ESS* jupyter notebook [31].

4.1. Classic statistical analysis

Before jumping into the machine learning algorithms, let us gain some intuition about the dataset we are working with and establish a baseline to which to compare the success of the subsequent implementation of machine learning analysis.

For each data entry we define the variable x as the average of all the components in the $X = \boldsymbol{\Phi}_E(\theta)$ tuple, that is $x = \langle X \rangle = \langle \boldsymbol{\Phi}_E \rangle$, and the variable y as the sum of the `pinslots` in Y, thus $y \equiv n$. If we plot these two variables against each other we obtain Fig. 9, where we can see a clear relationship between the two in a very characteristic leaf shape, as well as the discretisation of the possible values of n. We have also highlighted an arbitrary point in magenta to make the coming discussion easier to follow.

To characterise this dependency we wish to provide an interval of an estimator of n, in line with our desired confidence level, given a value of $\langle \boldsymbol{\Phi}_E \rangle$; which we have done by fitting a curve around the leaf. We have selected said curve to be a rotated ellipse, which was hypothesised by looking at the leaf's shape. To test its fit, we performed a few transformations on the data. We want to centre, rescale the abscissa axis to be comparable to the ordinate (for numerical reasons), and rotate the leaf so that its analytical description is simpler. All these transformations are summed up by writing the new coordinates $(x', y')^T$ in terms of the current ones $(x, y)^T$ as in the equation below.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}}_{45^\circ \text{ rotation}} \underbrace{\begin{pmatrix} m & 0 \\ 0 & 1 \end{pmatrix}}_{\text{Rescaling}} \underbrace{\left[\begin{pmatrix} x \\ y \end{pmatrix} - \bar{y} \begin{pmatrix} 1/m \\ 1 \end{pmatrix} \right]}_{\text{Centering the data}} = \begin{pmatrix} mx + y - 2\bar{y} \\ y - mx \end{pmatrix} , \quad (5)$$

where m is the slope of best fit between y and x and $\bar{y}$ is the average value of $y \equiv n$, which is ${}^{s^2}/_2 = 24.5$ in our case. Note that $\bar{x} = {}^{\bar{y}}/_m$, with $\bar{x}$ the expected average of the fluence ("expected" and not sample average to compensate the lack of $n = 0$ data entries), and that the point $(\bar{x}, \bar{y})$ describes the centre of the leaf before transformation, thus the centring operation. With a bit of thoughtful intuition, one can see that x' represents the position along the line of best fit and y' is the perpendicular distance to it. Note that once they are numerically comparable the scaling of the variables is of no importance for our analytical description, thus the missing $1/\sqrt{2}$ factor in the rotation. These new coordinates do not carry any physical meaning.

Once the data was conveniently transformed we proceeded to fit it to an ellipse, which as we know has equation:

$$\frac{(x')^2}{a^2} + \frac{(y')^2}{b^2} = 1 , \quad (6)$$

where a is the half-width and b is the half-height. We took $a = 2\bar{y} + \varepsilon$ with $\varepsilon = 0.01$ an offset, and b remained an adjustable parameter to have the ellipse include a different percentage of the data entries within its bounds. We consider the ellipse a good description of the leaf if, for all b, it maintains the same percentage of entries for all values of n. In this case, this percentage becomes our confidence level $\zeta(b)$, and b allows us to arrive to the width of the prediction interval of n.

Fig. 10 shows that the ellipse is a good fit, as the accuracies remain constant save statistical fluctuation (this variance decreases when the dataset size is increased).

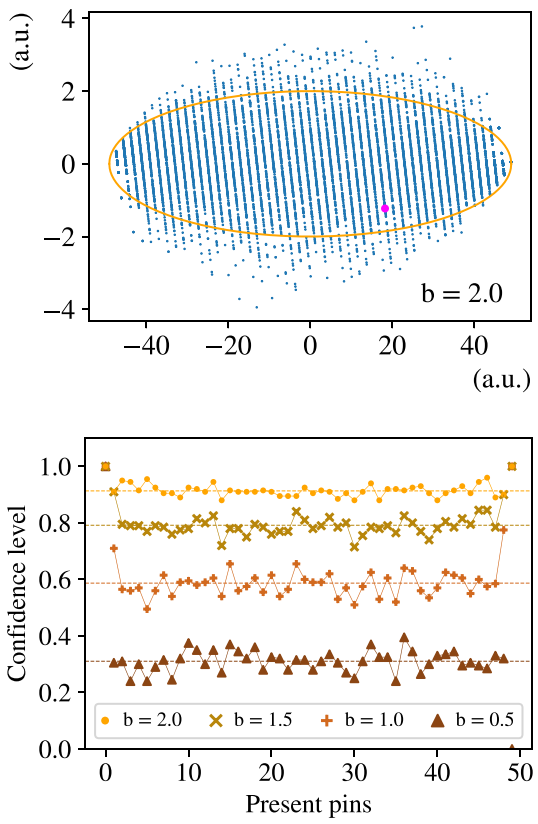

Fig. 10. Top: Ellipse fitted to the rotated leaf.
Bottom: Ellipse description of the data. Note how the percentage of data entries within the ellipse remains constant for all values of n (save statistical fluctuation), indicating a good fit. Their average (dotted line) becomes the confidence level ζ.

From here, we only need to determine the corresponding value of b given the desired confidence level, and apply the inverse transformation of Eq. (5) to the ellipse of Eq. (6). The former can be easily done by computing the overall confidence level for a set of values of b, $\zeta(b)$, and interpolating the inverse function $b(\zeta)$ (see Fig. 14). Once we have obtained b and if we denote our estimator interval by $\left(y_\zeta^-(x), y_\zeta^+(x)\right)$, the latter grants us the following expression.

$$y_\zeta^\pm(x) = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \,, \quad \text{with} \begin{cases} A = 4\bar{y}^2 + b^2 \,, \\ B = 2mx\left(b^2 - 4\bar{y}^2\right) - 4\bar{y}b^2 \,, \\ C = m^2x^2A - 4\bar{y}b^2\left(\bar{y} - mx\right) - 4\bar{y}^2b^2 \,. \end{cases} \tag{7}$$

Following this procedure, one obtains that for an average fluence $\langle \Phi_E \rangle = 0.50\ \gamma/\mathrm{s\,cm^2}\ 0.5$ keV, which as can be seen in Fig. 9 falls into the mid-low range of fluences, the prediction interval at 70% confidence is $(15.26, 17.64)$. Nevertheless, n must be a natural number, hence this prediction shows that the assembly has between 15 and 18 present pins. A more restrictive confidence level of 95% showcases this particular assembly with 14 to 19 pins.

4.2. Introduction of neural networks

For the neural network analysis we have used the $X = \Phi_E$ and $y \equiv n$ variables as defined above. Our first step was to normalise both variables and shuffle the entries, before splitting the dataset into training and validation sets in a four to one ratio (see Section 1.5). We used the npins accuracy function introduced earlier (see Section 3.3) for performance verification.

4.2.1. Implementing a DNN

After trying different configurations of dense neural networks following the design of Fig. 3, we achieved an optimal architecture with the following parameters. The input layer has $p = 32$ entries and the output is a scalar predicting n/s^2. There are two hidden layers with 100 and 30 neurons respectively with the exponential linear unit (elu) as the activation function (see Fig. 12). We have chosen the Adam optimiser [25] with a learning rate $\eta = 10^{-3}$, batch size 10 and 300 epochs. There is no reasoning behind the choice of these particular values other than being those which heuristically attain the highest npins accuracy.

The results of the training are displayed on the left of Fig. 11, where we can see the loss function in violet along with the npins accuracy in dark blue. We can see that the network has substantially improved the results provided by the statistical analysis, as it is correctly predicting the exact number of present pins for just under 70% of the data, compared to the four-pin wide interval we had obtained classically. There is also no evidence of overfitting, as the validation accuracy is similar to the training one.

However, there appears to be a performance threshold. This 70% accuracy is never overcome by the network, not if the epochs are increased nor if other hyperparameters are tweaked. It also appears when modifying the network architecture: adding or removing hidden layers, adjusting the number of neurons in each, or choosing different activation functions only alters the speed at which the network learns, but the bound remains at around 70%. Switching the optimiser to the SGD, which reduces the fluctuation of the fit in exchange for a considerably longer training time, is not of help either. The slower and more steady learning allows us to reach closer to the threshold, but it is indeed still there.

In regards to the other metrics, the optimal architecture for npins_accuracy provides average precision and recall scores very similar to the accuracy (both around 68%, differing in the decimal places). However, when studied individually for each class (each value of n) as described in Section 1.4, we see both metrics attain a perfect score for extreme n and decrease for middle values (see Fig. 13). This minimum is around 60% for both when discarding statistical fluctuations. One can see that this curved appearance of the metric was also present for the statistical analysis although to a much lesser extent (see Fig. 10), as the consistency of the metric was precisely the criterion we used to discern a good fit of the data. This shape will be discussed in Section 6.

4.2.2. Upgrading to a CNN

Recall that the strength of convolutional networks was their suitability for data with spatial correlation, which being Φ_E a continuous function of θ, is our case. Hence, the introduction of these structures is expected to improve the obtained results. The second network we have designed maintains the same 32-component vector entry and scalar output, but in this case, the hidden layers constitute a convolutional layer of 8 kernels of size 3 by 1 followed by a maxpooling layer, before being flattened and fed to a dense layer with 50 neurons. We maintain elu as the activation function for all neurons, and the optimiser is still Adam, but as the introduction of convolutional layers appeared to speed up the training we decreased the learning rate to $\eta = 10^{-4}$ for a finer minimisation of the loss function. Again the choice of these parameters was done heuristically. This yields us the results on the right of Fig. 11.

As was expected, the results slightly improve those of the DNN as we indeed surpassed 70%, but we still find ourselves limited by a very similar, slightly higher threshold. Precision and recall scores maintain the shape but also suffer a slight increase, as in Fig. 13. Changing the number and size of the layers, the activation functions or the hyperparameters still does not allow us to overcome the bound, and more epochs do not solve the problem either. However, it is evident that the implementation of convolutional layers enhances performance and slightly boosts training speed, probably due to the fewer amount of parameters.

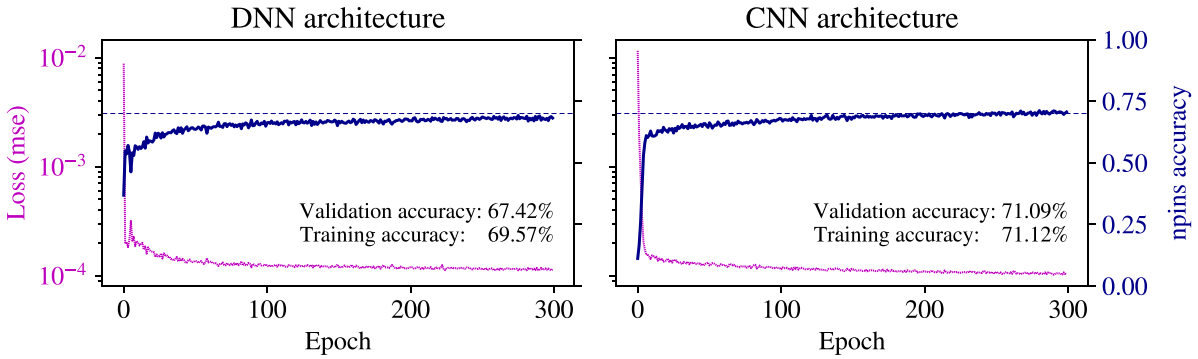

Fig. 11. Neural network training evolution for a 7 by 7 assembly. The dashed dark blue line corresponds to a 70% performance accuracy.

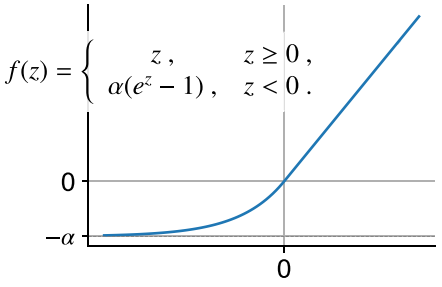

Fig. 12. Elu activation function.

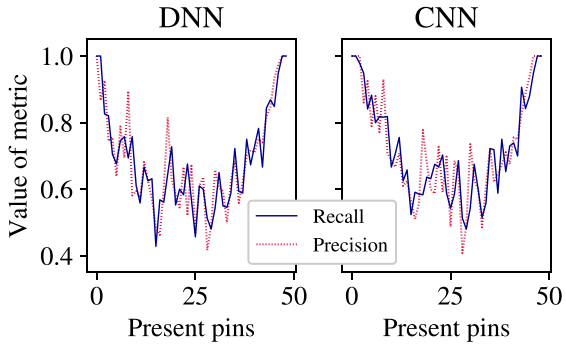

Fig. 13. Multiclass recall and precision for a 7 by 7 assembly. Evaluated on the validation set.

In an attempt to overcome the limit, we changed the value of l to determine whether this bound depends on the dataset size, as it is oftentimes said that the more data a neural network has available, the more it learns. We generated an additional 800 assembly data entries, which form our new training set, and the previously generated dataset with $l = 200$ became the validation set, representing a 500% increase in the total dataset size. Nonetheless, it was to no avail.

Despite all this, when adjusting the e parameter in npins accuracy we obtain that the predictions fall within ± 1 of the real value of n for the totality of the dataset, meaning that **all the unsuccessful predictions are off by only one pin (out of 49)**. This is an important improvement on the results achieved with the classical analysis.

5. Bigger assemblies

Let us scale up now. To begin with, we briefly discuss the extension of the statistical study to the new assembly sizes. Consider a leaf plot for an assembly size s and a confidence ζ at which we desire to provide a confidence interval $\left(y_\zeta^-(x), y_\zeta^+(x)\right)$ for an average fluence $x = \langle \Phi_E \rangle$. Then the confidence interval of maximum extension, $\left(y_\zeta^-, y_\zeta^+\right)$, can be

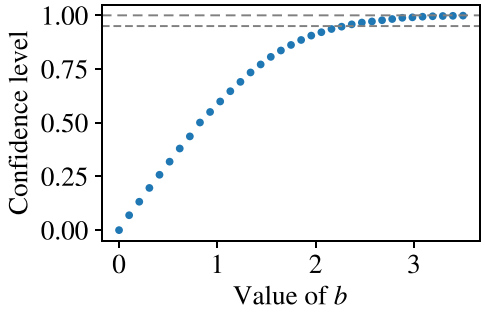

Fig. 14. Interpolation of b for the ellipse fit of the 7 by 7 assembly. The two dashed lines denote the 1 and the 0.95 confidence levels.

approximated with little error for non-extreme confidence levels ($\zeta < 0.95$) to be given by the following expression.

$$y_\zeta^\pm \approx \bar{y} \pm b(\zeta)\,, \tag{8}$$

where $\bar{y} = {}^{s^2}/_2$ and b is a parameter of the aforementioned ellipse (Eq. (6)).

Proof. Considering the geometric properties of an ellipse, the maximum length interval will occur exactly for the average fluence $\bar{x}$. Therefore, $y_\zeta^\pm = y_\zeta^\pm(\bar{x})$. Substituting in Eq. (7), and as $m\bar{x} = \bar{y}$, we obtain,

$$A = 4\bar{y}^2 + b^2\,,$$
$$B = 2m\bar{x}\left(b^2 - 4\bar{y}^2\right) - 4\bar{y}b^2 = -2\bar{y}\left(b^2 + 4\bar{y}^2\right) = -2\bar{y}A\,,$$
$$C = m^2\bar{x}^2 A - 4\bar{y}b^2\left(\bar{y} - m\bar{x}\right) - 4\bar{y}^2b^2 = \bar{y}^2\left(A - 4b^2\right)\,.$$

Thus

$$y_\zeta^\pm(x) = \frac{2\bar{y}A \pm \sqrt{16A\bar{y}^2b^2}}{2A} = \bar{y} \pm \frac{2\bar{y}b}{\sqrt{A}}\,.$$

Now, we know that $A = 4\bar{y}^2 + b^2$, and we can see in Fig. 10 that under transformation of Eq. (5), $2\bar{y} = a$ is approximately an order of magnitude higher than b. This difference is enhanced further when the squares are taken, making the b^2 term negligible when compared to $4\bar{y}$. In that case, $A \approx 4\bar{y}^2$ and thus we arrive at Eq. (8).

However, as can be seen in Fig. 14, b rapidly increases when we raise the desired confidence level above 0.95, inducing more error in the approximation. Thus our (arbitrary) choice of limiting the result to under 95% confidence. This similarly holds for the other assembly sizes. □

Therefore a representative enough comparison metric is the necessary value of b for a 70% confidence. This value ranges from 0.24 for the three by three assembly, through 1.26 for the already studied seven by seven, up to 4.00 for $s = 15$ and 4.89 for 17 by 17. These are considerably large, especially when considering the real-life assembly size $s = 17$. An interval of length almost 10 at 70% confidence, even

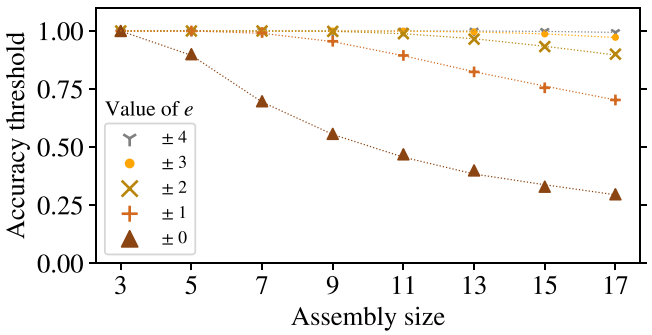

Fig. 15. Performance threshold for different values of the error e in npins accuracy. The markers represent the validation sets' accuracy, while the underlying dotted lines correspond to the training ones. The represented values have been computed using the CNN architecture.

out of 289 pins, is a very coarse prediction of n.

5.1. Neural network analysis

We have come to see working on 7 by 7 assemblies that there appears to exist a threshold to the performance of the network analysis shared by the two proposed architectures. In this case, for verification we fitted both to the datasets of all assembly sizes in consideration, $s \in \{3, 5, 7, \dots, 17\}$, and in all of them we find a threshold that similarly limits the performance for DNNs and CNNs, confirming our presumption. We call τ said threshold, and we are interested in studying its dependence on the assembly size and ultimately examining what we can do to overcome this hindrance. To do so it is useful to regard different values of e in the npins accuracy function (see Section 3.3). We write τ_e, with special interest in the exact classification accuracy τ_0.

τ_0 appears to have a sort of inverse proportionality relationship with s. This is quite unfortunate, as $\tau_0(17) < 30\%$ is not as handy for real-world applications as one would have hoped. In spite of this, $\tau_1(17) = 0.704$, meaning that at 70% confidence we are classifying at ± 1, which again represents an important improvement on the classical estimator interval. Moreover, the neural network allows us to classify with more than 99% confidence assemblies of size 17 by 17 if we are willing to take an uncertainty of 4 ($\tau_4(17) = 0.994$), which out of the 289 possible values of n represents only a 1.4% relative error.

Thus, even with the threshold, neural networks prove a powerful new tool in the prediction.

6. Overcoming the threshold

Despite the great improvements to the prediction we have been able to achieve so far, the existence of the threshold is somewhat bothersome. This value must surely be a theoretical constraint of the measuring system, especially given the smoothness of $\tau(s)$ seen in Fig. 15, which could serve to explain why no neural network architecture can overcome it.

6.1. Constraints of the current set-up

To begin with, let us briefly discuss why the introduction of neural networks improves the results of the statistical analysis. Trivially, it has to do with taking p entries, instead of only one, and the fact that those p entries change the angle under which we see the assembly, which allows the network to learn to distinguish between linearly distributed missing pins. Take for example the two assemblies on the left of Fig. 16: a detector at an angle $\pi/4$ could have trouble distinguishing between the two, as the above assembly is missing one very contributing pin (as

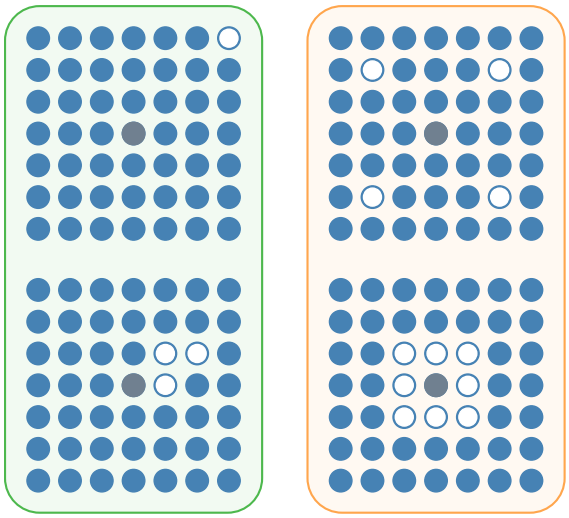

Fig. 16. Assembly samples for linear equivalency (left) and radial symmetry (right).

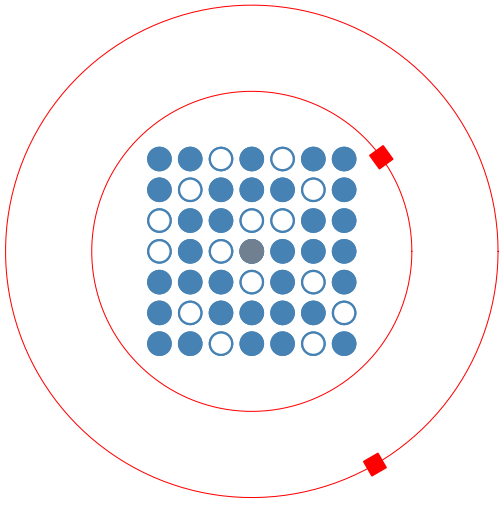

Fig. 17. Double detector set-up with $r_2 \approx \frac{3}{2} r_1$ around an assembly with high radial symmetry.

it is closer) to Φ, whereas the bottom one misses three less contributing ones. Simply circling the detector around the assembly shifts the weights of Eq. (3) and allows the algorithm to distinguish between the two.

Hence, it seems intuitive to consider possible constraints derived from the detector's radial symmetry to explain the threshold. Take now the two assemblies on the right of Fig. 16, which have radial symmetry in the missing pins. However, one has fewer missing further from the centre and the other has more missing in the middle. Again due to the way the pin contribution to Φ goes as the inverse of the distance squared, the Φ tuple for both assemblies may be very similar and thus the network could have a hard time distinguishing between the two cases. In fact, this is exactly what can be seen in the shape of the metrics in Fig. 13, as the number of possible combinations of radial symmetry in an assembly are maximal when the number of missing pins is near the half of the total, and decrease for extremal values of n. Now, this thought experiment gives us a very simple solution: if the problem is the countervailing between the distance and the number of the missing pins, it should be enough to add a second detector at a different radius to allow the network to see this offset between distance and number of missing pins.

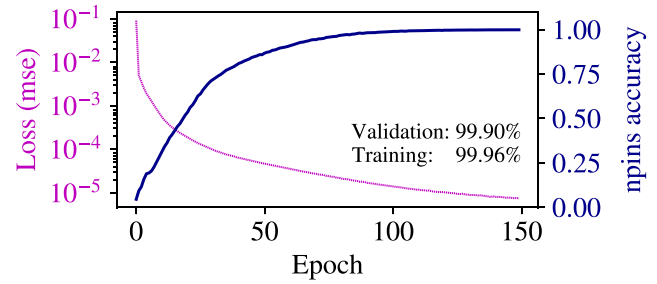

Fig. 18. CNN training evolution for a 7 by 7 assembly and a double detector set-up.

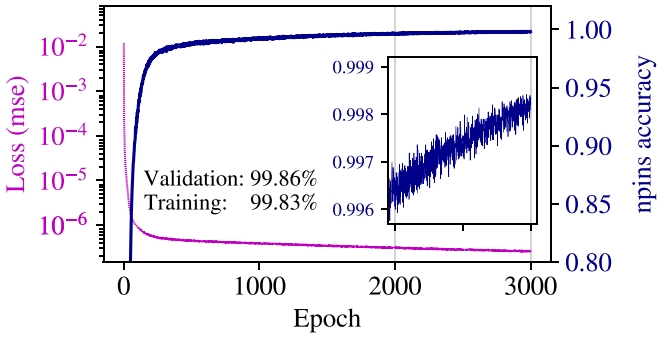

Fig. 19. CNN training evolution for a 17 by 17 assembly and a double detector set-up. Note the change in the accuracy range and the zoom window of the accuracy tendency of the last 1000 training epochs.

6.2. Introducing a second detector

It was in our best interest to place the first detector as close as physically possible to the assembly to maximise the variation with θ. When it comes to the second detector, we wanted it far enough from the first one so that the offset becomes negligible but not too far that the assembly appears as a single point to the detector, as we still want dependency with θ. The value we have used for the numerical calculations is $r_2 \approx \frac{3}{2} r_1$, but this was an arbitrary choice. See Fig. 17 for a schematic representation.

The data generation was similar to the previous case, the only difference being that now $X = \left(\boldsymbol{\Phi}_{E,r_1}, \boldsymbol{\Phi}_{E,r_2}\right)$, a two-tuple of vectors each of length p. For this we followed the procedure detailed in Section 3.4, but when copying *baseDet* we circled the assembly once at radius r_1, then changed it to r_2, and circled the assembly again at radius r_2. Otherwise no changes were made. The neural network analysis was also easily adaptable, especially with the CNN architecture, which was only altered to take two vectors as entries instead of only one. We decided against keeping using the DNN architecture as the CNN was both more easily adaptable and also attained higher accuracies, as seen above. In the same line as the previous case, we began with the seven by seven assembly as a first test of the results, which as it can be seen in Fig. 18 looks extremely promising.

Remark that the shown npins accuracy is for $e = 0$, which is to say that we are getting 99.90% correct predictions of the *exact* value of n (corresponding to just under 3.5σ). The precision and recall are both 1 for all n except for a few values, where they decrease to around 0.995, and in average they are both over 0.999. This network was trained using Adam, and we can see that the loss function seems to not have reached a minimum yet, which means that there does not appear to be any threshold: the learning does not become stagnant. Surely with more epochs, we could reach perfect prediction. The results for a 17 by 17 assembly are as shown in Fig. 19.

The performance speaks for itself. **Even at 99.86% prediction accuracy, all the remaining 0.14% assemblies which the network does not correctly predict are off by only one pin** ($R^2 = 0.999997$). Precision and recall are both over 0.9986 as well, and the dependency with n is not significative enough to suggest curvature. With the simple trick of adding a second detector we have been able to entirely surpass the machine learning threshold, and there is no evidence of the presence of another. We can see in the zoom of Fig. 19 that the tendency is increasing, meaning that **if we wanted a better performance a longer training time would suffice**.

This second set-up proposition is both reliable for real-world applications and practical, especially as the modification to implement the double detector does not necessarily need more investment than a second railing on which to slide the detector. Future work in this set-up would determine the dependence of the performance and the training speed with the dataset size l and the r_1 to r_2 ratio, as well as the necessary number of measurements p' of the second detector to break the symmetry, as we have taken here $p' = p$, but we anticipate that p' could be smaller than p.

7. Considering different spectra

After the success of our method under the assumption of equal pin spectra, we tested it with a more realistic assignment of fluence. For this, we have available three spent nuclear fuel spectra measured at the ORNL IFEL, which we denote by ϕ^1, ϕ^2, ϕ^3. We could have chosen to randomly assign one of the three to each pin, but to truly simulate different emission fluency for all the pins we have generated convex linear combinations of the three. Thus, the spectre that we will assign to rod k will be given by

$$\phi_k = \sum_{i=1}^{3} \lambda_{k,i} \phi^i \,, \qquad \text{with } \sum_{i=1}^{3} \lambda_{k,i} = 1 \ \text{ for all } k = 1, \ldots, s^2 \,. \tag{9}$$

It is worth mentioning that, given the fact that we are really only looking at the fluence of one value of energy and that all three spectra have quite similar values of it (the maximum relative difference is slightly over 9%), Eq. (9) can be considered as an elaborate way of adding noise to our data generation. Anyhow, whichever way we look at it, knowing how resistant the network is to these fluctuations is necessary for its characterisation. This variance can account for different burnup regions within an assembly, especially regarding individual pin burnup against overall assembly burnup. As an example, in Ref. [33] we can see that for uranium oxide reactors (which is the case of most PWR reactors) emission is almost linear with burnup, and as per [34] different pin burnup in a PWR assembly is shown to be within our 9% relative difference bound. Note that we have not organised burnup according to the distribution that appears in [34], but randomly instead, meaning that our spectra assignment is a generalisation of the real-world case.

7.1. Double detector

Let us jump straight into discussing the double detector configuration. In Fig. 20 we can see the performance results of the network's analysis on the 7 by 7 assembly with different spectra, which are almost identical to those of equal spectra in Fig. 18. However, the network fits more slowly: to arrive at the same performance we need twice as many epochs. In regards to precision and recall, they are both as high as the accuracy. However, it is interesting to note that the few incorrectly predicted assemblies tend to accumulate towards high values of n, instead of describing a centred curve as we saw for the single detector set-up. This is unsurprising as those are the assemblies whose collective emissions $\boldsymbol{\Phi}_E(\theta)$ vary most with the introduction of convex pin spectra. Additionally, similarly to what we observed for Fig. 18, the network does not seem to have a performance bound (note the loss function's decreasing tendency), so we can deduce that our method is capable of perfectly distinguishing single pin differences in 7 by 7 assemblies.

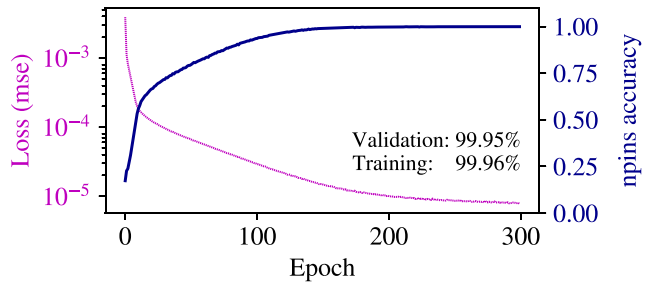

Fig. 20. CNN training evolution for a 7 by 7 assembly with different spectra assignment and a double detector set-up.

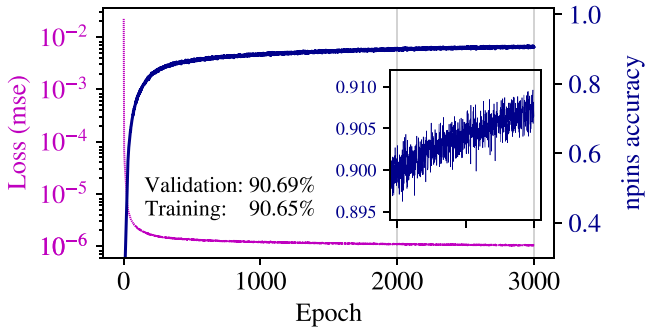

Fig. 21. CNN training evolution for a 17 by 17 assembly with different spectra assignment and a double detector set-up. Note the change in the accuracy range with respect to Fig. 19.

In the case of 17 by 17 assemblies, we observe in Fig. 21 an even slower learning process, which was to be expected due to the considerable increase in size. Despite this, we chose to maintain the network's hyperparameters the same as in the equal spectra case: a learning rate of $\eta = 10^{-5}$ and 3000 epochs, for ease of comparison with Fig. 19.

Remarkably, even with the introduction of different spectra, the network is capable of attaining over a 90% exact prediction accuracy with an increasing tendency. Additionally, the npins accuracy result for a ± 1 classification of the validation set is 100% (the R^2 score of the network is 0.999988). The zoom of Fig. 21 shows that, although slow-going, the accuracy lacks any threshold and that its arrival to 99% accuracy would only need further training.

This training could simply entail more epochs, but we suggest for future work trying other neural network fit optimisation methods, which we know is a research field in itself and falls outside our area of expertise. We believe our results could benefit from using the BFGS optimiser [35], which as shown in [36] can be used after an initial training with Adam to considerably decrease the final value of the mse, at the cost of a more demanding run.

As for the multiclass recall and precision metrics, they are represented in Fig. 22. We can see that both performance metrics are clearly decreasing with the number of present pins, which is in line with what we are to expect after the introduction of the convex spectra, as mentioned previously. Further work would study these metrics' dependency with the intensity of the noise introduced by Eq. (9), and set data noise comparable to real world values of single pin emission variance for collectively burned pins (recall that our dataset had up to a 9% relative difference between spectra).

Nonetheless, we are convinced the presence of noise gives more importance to the dataset size. To verify this, we have run a thousand epochs of training with the same network architecture on the current dataset randomly cut in half (corresponding to $l = 100$), and into a quarter ($l = 50$). We obtain accuracies of 88.01% and 83.72% respectively, compared to the 88.59% of the $l = 200$ dataset at a

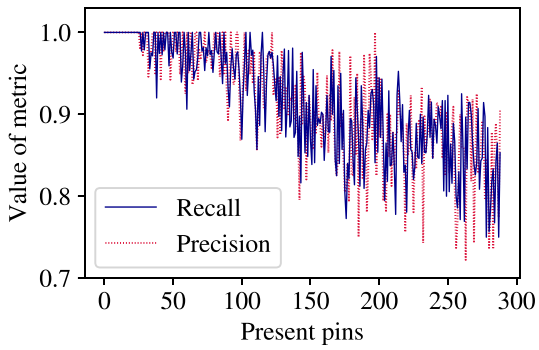

Fig. 22. Multiclass recall and precision for a 17 by 17 case with convex spectra and double detector set up. Evaluated on the validation set.

thousand epochs. This hints at the influence of the dataset size, with no evidence of the presence of a bound to the maximum performance the network can reach, especially considering the impressive accuracy for the 7 by 7 assembly; only that the learning process is more demanding.

Hence, we hypothesise that a similarly high accuracy can be attained for a 17 by 17 assembly, given enough computational capacity, as both the data generation and the network training process would benefit from it.

8. Conclusions

We have proved the outstanding performance of the proposed neural network for the blind characterisation of nuclear fuel assemblies, capable of determining the exact number of pins present in a full-size PWR fuel assembly, even when considering different emission spectra for each pin. Neural networks have an exceptional capacity to accurately predict the specifics of a given PWR fuel assembly. It is consequently reasonably expected to carry this performance over to fuel assemblies of other types of reactors. The prediction capacity reaches over 3.5σ with a truly simple architecture thanks to the proposed double detector set-up introduced in this article. This new tool has the potential to help in spent nuclear fuel safeguarding efforts by providing an accessible, fast, non-destructive method to blindly and inequivocally determine the burnt uranium content in a nuclear assembly.

For future work, we propose the use of Geant4 [37] to consider attenuation of gamma rays through the nuclear assembly. The scope of this article is not targeting attenuation, but rather development and proof of concept of a machine learning tool for the characterisation of nuclear assemblies. However, we expect that the inclusion of attenuation will enhance the difference of contribution of each pin through its greater dependency with its location and with the detector's placement. We anticipate that determining the Pu-U ratio through XRF for the whole assembly will be challenging since a greater angular resolution of the measured spectrum would be necessary due to self-attenuation.

To finalise, we would like to propose a measuring campaign at ORNL or any similar reprocessing nuclear facility in Europe to validate our studies in the field and deal with real data from the assemblies.

CRediT authorship contribution statement

J. Paz-Peñuelas-Oliván: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **J. Ruz:** Writing – review & editing, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to extend our gratitude to the Lawrence Livermore National Laboratory, the Oak Ridge National Laboratory, the Los Alamos National Laboratory and the Idaho National Laboratory for providing the three spectra that sparked this project and encouraging us to publish its promising results. Spectral data used for this work was collected and published in [19] under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory [38] under contract DE-AC52-07NA27344, by Oak Ridge National Laboratory [39] under contract DE-AC05-00OR22725, by Los Alamos National Laboratory [40] under contract DE-AC52-06NA25396 and by Idaho National Laboratory [41] under contract DE-AC07-05ID14517.

References

[1] Alfred Strasser, et al., Fuel Fabrication Process Handbook, Revision 1, Advanced Nuclear Technology (A.N.T.) International, Analysvägen 5, SE-435 33 Mölnlycke, Sweden, 2014.
[2] Alissa Sarah Stafford, Spent Nuclear Fuel Self-Induced XRF to Predict Pu to U Content (M.Sc. thesis), Texas A&M University, TX, 2010.
[3] IAEA safeguards glossary, International Nuclear Verification Series, (3 (Rev. 1)) INTERNATIONAL ATOMIC ENERGY AGENCY, Vienna, 2022.
[4] World Nuclear Association, Plutonium, 2023, https://world-nuclear.org/information-library/nuclear-fuel-cycle/fuel-recycling/plutonium#:~:text=The%20approximately%201.15%25%20of%20plutonium,source%20of%20heat%20and%20radioactivity.
[5] Shengli Chen, et al., Linear regression and machine learning for nuclear forensics of spent fuel from six types of nuclear reactors, Phys. Rev. Appl. 19 (2023) 034028.
[6] Zhangpeng Guo, et al., Defect detection of nuclear fuel assembly based on deep neural network, Ann. Nucl. Energy 137 (2020) 107078.
[7] Arnau Albà, et al., Fast uncertainty quantification of spent nuclear fuel with neural networks, Ann. Nucl. Energy 196 (2024) 110204.
[8] Bassam A. Khuwaileh, Belal Almomani, A once-through artificial neural network approach for used nuclear fuel inverse depletion analysis: A comparative study, Ann. Nucl. Energy 205 (2024) 110598.
[9] Bamidele Ebiwonjumi, et al., Machine learning of LWR spent nuclear fuel assembly decay heat measurements, Nucl. Eng. Technol. 53 (11) (2021) 3563–3579.
[10] Riccardo Rossa, et al., Comparison of machine learning models for the detection of partial defects in spent nuclear fuel, Ann. Nucl. Energy 147 (2020) 107680.
[11] Jin Whan Bae, et al., Deep learning approach to nuclear fuel transmutation in a fuel cycle simulator, Ann. Nucl. Energy 139 (2020) 107230.
[12] Hasan Özdoğan, et al., Estimations of level density parameters by using artificial neural network for phenomenological level density models, Appl. Radiat. Isot. 169 (2021) 109583.
[13] Hasan Özdoğan, et al., Estimations of giant dipole resonance parameters using artificial neural network, Appl. Radiat. Isot. 169 (2021) 109581.
[14] H. Özdoğan, et al., Mass excess estimations using artificial neural networks, Appl. Radiat. Isot. 184 (2022) 110162.
[15] Hasan Özdoğan, et al., Estimations for (n,α) reaction cross sections at around 14.5 MeV using levenberg-marquardt algorithm-based artificial neural network, Appl. Radiat. Isot. 192 (2023) 110609.
[16] Hasan Özdoğan, et al., Neural network predictions of (α,n) reaction cross sections at 18.5±3 MeV using the Levenberg-Marquardt algorithm, Appl. Radiat. Isot. 204 (2024) 111115.
[17] J. Ruz, Using X-ray optics for nuclear physics applications, CAPA, University of Zaragoza, Lawrence Livermore National Laboratory, 2022, SATURNALIA '22.
[18] J. Ruz, et al., Direct measurement of ^{235}U in spent fuel rods with Gamma-ray mirrors, Nucl. Instrum. Methods Phys. Res. A 777 (2015) 15–19.
[19] J. Ruz, et al., Characterization and simulation of soft gamma-ray mirrors for their use with spent fuel rods at reprocessing facilities, Appl. Opt. 55 (16) (2016) 4285–4292.
[20] K.P. Ziock, et al., Non-Destructive Assay of Spent Nuclear Fuel Using Gamma-Ray Mirrors as a Narrow Band Pass Filter, Institute of nuclear materials management (INMM), 2023.
[21] Michael A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.
[22] F. Rosenblatt, Principles of neurodynamics: Perceptrons and the theory of brain mechanisms, in: Cornell Aeronautical Laboratory. Report no. VG-1196-G-8, Spartan Books, 1962.
[23] Allan Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195.
[24] Elizabeth Million, The Hadamard product, 2019.
[25] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2017.
[26] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas Schön, Machine learning: A first course for engineers and scientists, 2022.
[27] Python programming language, https://www.python.org.
[28] Google, Tensorflow library with Keras API https://www.tensorflow.org/guide/keras.
[29] Scikit-learn python library, https://scikit-learn.org/stable/.
[30] Project Jupyter https://jupyter.org.
[31] Jorge Paz-Peñuelas Oliván, Nuclear Assemblies python library, https://github.com/JorgikNoob/Nuclear-Assemblies.
[32] GitHub developer platform, https://github.com.
[33] Impact of high burnup uranium oxide and mixed uranium-plutonium oxide water reactor fuel on spent fuel management, in: Nuclear Energy Series, (NF-T-3.8) International Atomic Energy Agency, Vienna, 2011.
[34] Radial Variation of Burnup and Source Terms in High Burnup LWR Fuel, Technical report, Oak Ridge National Laboratory, Oak Ridge, TN, 2016, pp. 1–6.
[35] Jorge Nocedal, Stephen J. Wright, Quasi-Newton methods, in: Numerical Optimization, Springer New York, New York, NY, 2006, pp. 135–163.
[36] Iván Alcaraz Aguado, Modelización de sistemas físicos y aproximación de EDPs mediante PINN's (Physics-Informed Neural Networks) (B.Sc. thesis), Universidad de Alicante, Alicante, 2024.
[37] CERN, Geant4 https://geant4.web.cern.ch.
[38] Lawrence Livermore National Laboratory, https://www.llnl.gov.
[39] Oak Ridge National Laboratory, https://www.ornl.gov.
[40] Los Alamos National Laboratory, https://www.lanl.gov.
[41] Idaho National Laboratory, https://www.inl.gov.