Supplemental of "Machine Learning of Implicit Combinatorial Rules in Mechanical Metamaterials"

Ryan van Mastrigt,[1,2,*] Marjolein Dijkstra,[3] Martin van Hecke,[2,4] and Corentin Coulais[1]

[1] *Institute of Physics, Universiteit van Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands*
[2] *AMOLF, Science Park 104, 1098 XG Amsterdam, The Netherlands*
[3] *Soft Condensed Matter, Debye Institute for Nanomaterials Science, Department of Physics, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands*
[4] *Huygens-Kamerling Onnes Lab, Universiteit Leiden, Postbus 9504, 2300 RA Leiden, The Netherlands*

(Dated: September 15, 2022)

Floppy and frustrated structures

In this section, we discuss in more detail the metamaterial M1 of Fig. 1. We first derive the design rules that lead to floppy structures, then we discuss the rarity of such structures.

Design rules for floppy structures

Here we provide a brief overview of the rules that lead to floppy structures for the combinatorial metamaterial M1 of Fig. 1. The three-dimensional building block of this metamaterial can deform in one way that does not stretch any of the bonds: it has one zero mode (see [1] for details of the unit cell). In two dimensions, there are two orientations of the building block that deform differently in-plane. We label these two orientations as green/red and white (Fig. 1(a)).

We can formulate a set of rules for configurations of these building blocks in two dimensions. Configurations of only green/red building blocks or white building blocks deform compatibly (C): the configuration is floppy. A single horizontal or vertical line of white building blocks in a configuration filled with green/red building blocks also deforms compatibly. More lines (horizontal or vertical) of white blocks in a configuration filled with green/red blocks deform compatibly if the building block at the intersection of the lines is of type green/red (Fig. 1(b)).

In summary, we can formulate a set of rules:

i All white building blocks need to be part of a horizontal or vertical line of white building blocks.

ii At the intersection of horizontal and vertical lines of white building blocks there needs to be a green/red building block.

If these rules are met in a configuration, the configuration will be floppy (C). A single change of building block is sufficient to break the rules, creating an incompatible (I) frustrated configuration (Fig. 1(b)).

Rarity of floppy structures

Here we show how the rarity of class C depends on the size of the $k_x \times k_y$ configuration. To show this, we simulate configurations with varying $k_x, k_y \in \{2, 3, 4, 5, 6\}$. The size of the design space grows exponentially as $2^{k_x k_y}$, yet the fraction of class C configurations decreases exponentially with unit cell size (Fig. A1). Thus the number of C configuration scale with unit cell size at a much slower rate than the number of total configurations. For large configuration size, the number of C configurations is too small to create a sufficiently large class-balanced training set to train neural networks on.

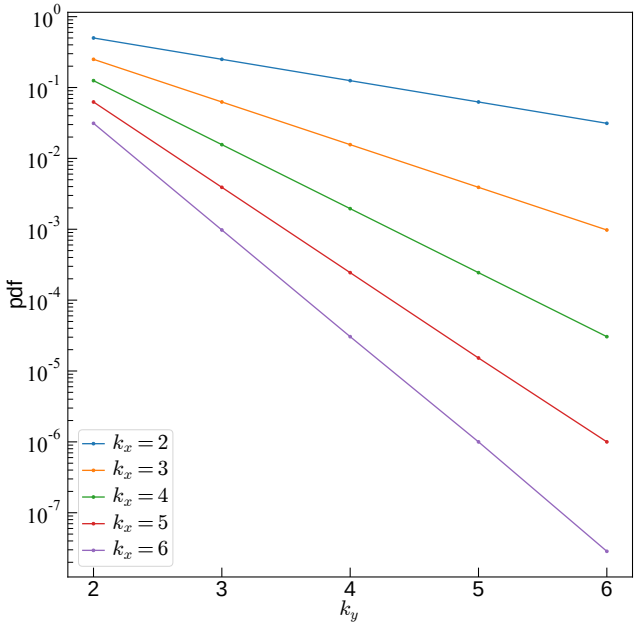

Figure A1. Probability density function (pdf) of $k_x \times k_y$ class C configurations.

Zero modes in combinatorial metamaterials

In this section, we present theoretical and numerical results at the root of the classification of zero modes in the combinatorial metamaterial M2 of Fig. 2. We first derive the zero modes of the building block, then we postulate a set of rules for classification (i) of unit cells. Finally, we provide numerical proof of those rules.

Zero Modes of the Building Block

The fundamental building block is shown schematically in Fig. A2. Each black line represents a rigid bar, while vertices can be thought of as hinges; the 11 bars are free to rotate about the 8 hinges in 2 dimensions. The colored triangles form rigid structures, *i.e.* they will not deform. From the Maxwell counting [2] we obtain $N_{zm} = 2 \cdot 8 - 11 - 3 = 2$, where the 3 trivial zero modes in 2 dimensions, translation and rotation, are subtracted such that N_{zm} is the number of zero modes of the building block. The precise deformation of these two zero modes can be derived from the geometric constraints of the building block.

To derive the zero modes to linear order, we note that they preserve the length of all bars, such that the modes can be characterized by the hinging angles of the bar. Let A, B, C, D, and E denote these angles. Going around the loop $ABCDE$, the angles add up to 3π:

$$A + B + C + D + E = 3\pi. \tag{A1}$$

Next, we expand the angles from their rest position to linear order:

$$\begin{aligned} A &= \tfrac{\pi}{2} + \alpha,\ B = \tfrac{3\pi}{4} + \beta,\ C = \tfrac{\pi}{2} + \gamma, \\ D &= \tfrac{\pi}{2} + \delta,\ E = \tfrac{3\pi}{4} + \epsilon. \end{aligned} \tag{A2}$$

Then, from the condition that the bars cannot change length, we obtain

$$1 - \cos(A) = 3 - 2\cos(C) - 2\cos(D) + 2\cos(C + D), \tag{A3}$$

and

$$\sin(D) - \frac{\sin(D+E)}{\sqrt{2}} = \sin(C) - \frac{\sin(C+B)}{\sqrt{2}}. \tag{A4}$$

Up to first order in $\alpha, \beta, \gamma, \delta, \epsilon$, equations (A3) and (A4) can be rewritten as:

$$\alpha = 2\gamma + 2\delta, \tag{A5}$$

$$\delta + \epsilon = \beta + \gamma. \tag{A6}$$

Together with the loop condition (A1), we obtain a set of three equations which express α, δ and ϵ in β and γ:

$$\begin{pmatrix} \alpha \\ \delta \\ \epsilon \end{pmatrix} = \begin{pmatrix} -2 & -2 \\ -1 & -2 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} \beta \\ \gamma \end{pmatrix}. \tag{A7}$$

This demonstrates that we can choose the two parameters β and γ arbitrarily, while still satisfying equations (A1), (A5) and (A6), consistent with the presence of two zero modes.

We now choose the basis of the zero modes such that the first zero mode is the deformation of the square BCDE, such that $\alpha = 0$. This leads to the well-known counter-rotating squares (CRS) mode [3, 4] when tiling building blocks together. Thus we choose the basis

$$\begin{pmatrix} \beta \\ \gamma \end{pmatrix} = M_{CRS} \begin{pmatrix} -1 \\ 1 \end{pmatrix} + M_D \begin{pmatrix} 3 \\ -1 \end{pmatrix}. \tag{A8}$$

M_{CRS} is the amplitude for the counter-rotating squares mode, while M_D is the amplitude of the mode that does change corner A. We refer to this mode as the diagonal mode.

By tiling together the building block in different orientations, we can create 4^{k^2} size $k \times k$ unit cells. These unit cells — and metamaterials built from them — may have more or less zero modes than the constituent building blocks, depending on the number of states of self-stress. Previous work on 2×2 unit cells showed that each unit cell could be classified based on the number of zero modes [5]. Here, we consider the previously unexplored cases of 3×3 up to 8×8 square unit cells.

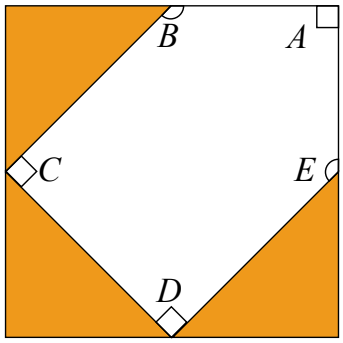

Figure A2. Schematic real space representation of the building block. A, B, C, D, and E label the five corners that can change angle under zero-energy deformations.

Rule-based classification of unit cells

Unit cells are classified based on the number of zero modes $M(n)$ for $n \geq 2$ as either class I or class C as described in the main text. Here we formulate a set of empirical rules that distinguishes class I unit cells from class C unit cells for classification (i).

Any finite configuration of building blocks, no matter the orientation of each block, supports the counter-rotating squares (CRS) mode with open boundary conditions, where all building blocks will deform with $M_{CRS} \neq 0$ and $M_D = 0$. They must all have equal magnitude $|M_{CRS}|$, but alternate in sign from building block to building block in a checkerboard pattern, similar to the ground state of the anti-ferromagnetic Ising model on a square lattice. An arbitrary configuration in the real space representation, and the CRS mode of that configuration in the directed graph representation, are shown in Fig. A3(a).

However, precisely because the building block supports another mode, there could in principle be other collective modes than the CRS mode in any given configuration. We have observed that class C unit cells have a specific structure, which we refer to as a strip mode. A strip mode spans the unit cell periodically in one direction, such that the total number of zero modes for a configuration of $n \times n$ tiled unit cells grows linearly with n.

The pattern of deformations for these modes consists of two rectangular patches of building blocks with CRS modes (where $M_D = 0$ for every building block) — potentially of different amplitude — separated by a strip of building blocks (the *strip*) that connects these patches, where $M_D \neq 0$. A unit cell configuration with a strip mode, which consists of building blocks in a strip of block-width $W = 2$ that deform with $M_D \neq 0$, and building blocks in the two areas outside of the strip, U & V, that do not deform, is shown in Fig. A3(a). Note that the CRS mode can always be freely added or subtracted from the total configuration.

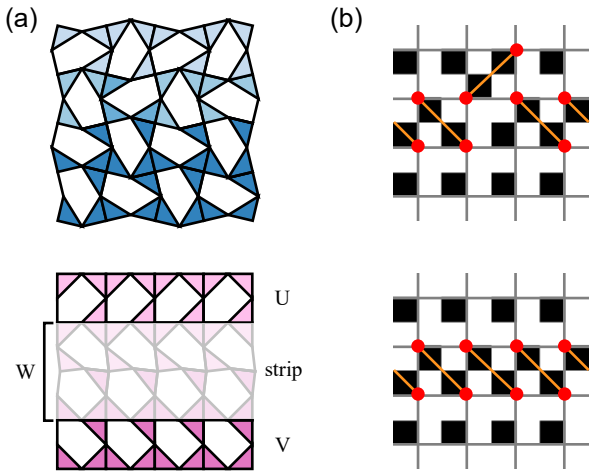

Figure A3. Schematic and pixel representation of modes in a 4×4 unit cell. (a) Schematic deformation of counter-rotating squares mode (top unit cell, blue) and a strip mode (bottom unit cell, pink). The strip mode spans the entire area of the strip (white) of width $W = 2$, while the areas U and V do not deform. (b) Respective pixel representations of the left unit cells. Paired unit cells are highlighted through red dots connected by orange lines. Note that the top unit cell does not contain a strip that meets the strip mode rules, while the bottom unit cell does.

i We conjecture that the presence of a strip mode is a necessary and sufficient condition for a unit cell to be of class C.

We verify (i) below. Moreover, we now conjecture a set of necessary and sufficient conditions on the configuration of the strip that lead to a strip mode. Underlying this set of conditions is the notion of *paired* building blocks: neighboring blocks that connect with their respective A corners, or equivalently, blocks that have their black pixels in the same plaquette in the pixel representation, see Fig. A3(b). Depending on the orientation of the paired building blocks, pairs of these blocks are referred to as horizontal, vertical or diagonal pairs. The set of conditions to be met within the strip to have a horizontal (vertical) strip mode can be stated as follows:

ii Each building block in the strip is paired with a single other neighboring building block in the strip.

iii Apart from horizontal (vertical) pairs, there can be either vertical (horizontal) or diagonal pairs within two adjacent rows (columns) in the strip, never both.

Consider the unit cells of Fig. A3, the top unit cell has multiple paired building blocks, but contains no horizontal (or vertical) strip where every block is paired. Conversely, the bottom unit cell does contain a strip of width $W = 2$ blocks where every block is paired to another block in the strip. Consequently, the bottom unit cell obeys the rules and supports a strip mode, while the top unit cell does not.

Each indivisible strip of building blocks for which these conditions hold, supports a strip mode. For example, if a unit cell contains a strip of width $W = 2$ which obeys the rules, but this strip can be divided into two strips of width $W = 1$ that each obey the rules, then the width $W = 2$ strip supports two strip modes, not one.

We refer to (i) as the strip mode conjecture, and (ii) and (iii) as the strip mode rules. We now present numerical evidence that supports these rules.

Numerical evidence for strip mode rules

The conjecture and rules (i)-(iii) stated in the previous section can be substantiated through numerical simulation. To do so, we determine the class of randomly picked unit cells.

To assess the rules, a large number of square unit cells are randomly generated over a range of sizes $k \in \{3, 4, 5, 6, 7, 8\}$. For each unit cell configuration, $n_x \times n_y$

metamaterials, composed by tiling of the unit cells, are generated over a range of $n_x = n_y = n \in \{1, 2, 3, 4\}$ for $k \leq 4$. From $k \geq 6$ onward, the 1×1 configuration is generated, as well as $n_x \times 2$ and $2 \times n_y$ configurations with $n_x, n_y \in \{2, 3, 4\}$ to save computation time.

The rigidity, or compatibility, matrix R is constructed for each of these configurations, subsequently rank-revealing QR factorization is used to determine the dimension of the kernel of R. This dimension is equivalent to the number of zero modes of the configuration, $M(n)$ is then equal to this number minus the number of trivial zero modes: two translations and one rotation.

From the behavior of $M(n)$ as a function of n, we define the two classes: I and C. In Class I $M(n)$ saturates to a constant for $n \geq 2$, thus class I unit cells do not contain any strip modes. Note that they could still contain additional zero modes besides the CRS mode. In Class C $M(n)$ grows linearly with n for $n \geq 2$, therefore class C unit cells could support a strip mode [6]. Moreover, if conjecture (i) is true, the number of strip modes supported in the class C configuration should be equivalent to the slope of $M(n)$ from $n \geq 2$ onward.

In class I, $M(n)$ is constant for sufficiently large n, thus class I unit cells do not contain any strip modes. Note that they could still contain additional zero modes besides the CRS mode. In class C $M(n)$ grows linearly with n for sufficiently large n, therefore class C unit cells could support a strip mode. Moreover, if conjecture (i) is true, the number of strip modes supported in the class C configuration should be equivalent to the slope of $M(n)$ for sufficiently large n.

To test conjecture (i) and the strip mode rules (ii) and (iii), we check for each generated unit cell if it contains a strip that obeys the strip mode rules. This check can be performed using simple matrix operations and checks [7]. If (ii)-(iii) are correct, the number of indivisible strips that obey the rules within the unit cell should be equal to the slope of $M(n)$ for class C unit cells, and there should be no strips that obey the rules in class I unit cells. Simulations of all possible $k = 3$ unit cells, one million $k = 4, 5, 6$ unit cells, two million $k = 7$ unit cells, and 1.52 million $k = 8$ unit cells show perfect agreement with the strip mode rules for unit cells belonging to either class I or C, see Fig. A4. Consequently, numerical simulations provide strong evidence that the strip mode rules as stated are correct.

Constructing and Training Convolutional Neural Networks for metamaterials

In this section, we describe in detail how we construct and train our convolutional neural networks (CNNs) for classifying unit cells into class I and C. We first transform our unit cells to a CNN input, secondly we establish the architecture of our CNNs. Next, we obtain the training set, and finally we train our CNNs.

Pixel Representation

To feed our design to a neural network, we need to choose a representation a neural network can understand. Since we aim to use convolutional neural networks, this representation needs to be a two-dimensional image. For our classification problem, the presence or absence of a zero mode ultimately depends on compatible deformations between neighboring building blocks. As such, the representation we choose should allow for an easy identification of the interaction between neighbors.

In addition to being translation invariant, the classification is rotation invariant. While we do not hard code this symmetry in the convolutional neural network, we do choose a representation where rotating the unit cell should still yield a correct classification. For example, this excludes a representation where each building block is simply labeled by a number corresponding to its orientation. For such a representation, rotating the design without changing the numbers results in a different interplay between the numbers than for the original design. Thus we cannot expect a network to correctly classify the rotated design.

For both metamaterials, we introduce a *pixel* representation. We represent the two building blocks of metamaterial featured in Fig. 1 as either a black pixel (1) or a white pixel (0) (Fig. A5(a)). A $k_x \times k_y$ unit cell thus turns into a $k_x \times k_y$ black-and-white image.

Likewise, we introduce a *pixel* representation for the metamaterial M2 of Fig. 2 which naturally captures the spatial orientation of the building blocks, and emphasizes the interaction with neighboring building blocks. In this

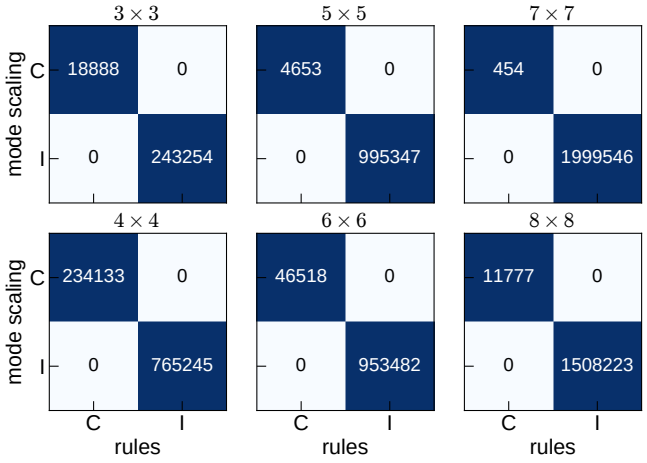

Figure A4. Confusion matrices for classification based on mode scaling in comparison to classification based on rules (i)-(ii). The $k \times k$ unit cell size is indicated on top of each matrix.

representation, each building block is represented as a 2×2 matrix, with one black pixel (1) and three white (0) pixels, see Fig. A5(b). The black pixel is located in the quadrant where in the bars-and-hinges representation the missing diagonal bar is. Equivalently, this is the quadrant where in the directed graph representation the diagonal edge is located. Moreover, in terms of mechanics, this quadrant can be considered floppy, while the three others are rigid.

This representation naturally divides the building blocks into 2×2 *plaquettes* in which paired building blocks are easily identified, see Fig. A5(b). Building blocks sharing their black pixel in the same plaquette are necessarily paired, and thus allow for deformations beyond the counter-rotating squares mode. Note that this includes diagonally paired building blocks as well. By setting the stride of the first convolution layer to $(2,2)$, the filters only convolve over the plaquettes and not the building blocks, which do not contain any extra information for classification.

CNN architecture details

To classify the unit cells into class I and C, we use a convolutional neural network (CNN) architecture. We first discuss the architectures used to obtain the results of Tab. 1. Then we discuss the architecture used to obtain the results of Fig. 4.

For the metamaterial M1 of Fig. 1, the CNN consists of a single convolution layer with 20 2×2 filters with bias and ReLu activation function. The filters move across the input image with stride $(1,1)$ such that all building block interactions are considered. Subsequently the feature maps are flattened and fully-connected to a hidden layer of 100 neurons with bias and ReLu activation function. This layer subsequently connected to 2 output neurons corresponding to C and I with bias and softmax activiation function. The input image is not padded. Since a network of this size was already able to achieve perfect performance, we saw no reason to go to a bigger network.

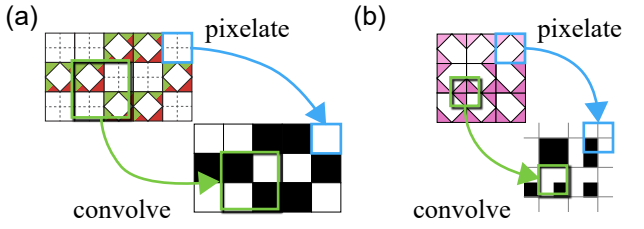

Figure A5. Unit cell designs of the combinatorial metamaterials in Fig. 1 (a) and Fig. 2 (b) and their respective pixel representations. The blue squares indicates how the building blocks are transformed to pixels, the green squares show which part of the unit cell is convolved by the first convolution layer.

For the metamaterial M2 of Fig. 2 and classification problem (i) we first periodically pad the input image with a pixel-wide layer, such that a $2k\times 2k$ image becomes a $2k+2\times 2k+2$ image. This image is then fed to a convolutional layer, consisting of 20 2×2 filters with bias and ReLu activation function. The filters move across the input image with stride $(2,2)$, such that the filters always look at the parts of the image showing the interactions between four building blocks (Fig. A5(b)). Subsequently the 20 $k+1\times k+1$ feature maps are flattened and fully-connected to a hidden layer of 100 neurons with bias and ReLu activation function. This layer is then fully-connected to 2 output neurons corresponding to the two classes with bias and softmax activation function. From the hyperparameter grid search (see section *CNN hyperparameter grid search details*) we noted that this n_f and n_h were sufficiently large for good performance.

For classification (ii) we again pad the input image with a pixel-wide layer. The CNN now consists of three sequential convolutional layers of increasing sizes 20, 80, and 160 filters with bias and ReLu activation function. The first convolution layer moves with stride $(2,2)$. The feature maps after the last convolutional layer are flattened and fully-connected to a hidden layer with 1000 neurons with bias and ReLu activation function. This layer is fully-connected to two output neurons with bias and softmax activation function. This network is larger than for classification (i); we saw noticeable improvements over the validation set when we considered larger networks. This is most likely a result of the (unknown) rules behind classification (ii) being more complex.

The networks are trained using a cross-entropy loss function. This loss function is minimized using the Adam optimization algorithm [8]. This algorithm introduces additional parameters to set before training compared to stochastic gradient descent. We keep all algorithm-specific parameters as standard ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1\mathrm{e}-07$), and only vary the learning rate η from run to run. The network for the classification problem of Fig. 1 uses a weighted cross-entropy loss function, where examples of C are weighted by a factor 200 more than examples of I.

To obtain the results of Fig. 4, we use the architecture of classification (i) and vary the number of neurons n_h in the hidden layer. We keep the number of filters the same. To obtain this architecture, we performed a hyperparameter grid search, where we varied the number of filters n_f of the convolution layer and the learning rate η as well. The details are discussed in the section *CNN hyperparameter grid search details*. The total number of parameters for this network with n_f filters and n_h neurons is

$$(4+1)n_f + ((k+1)^2 n_f + 1)n_h + (n_h+1)2. \qquad \text{(A9)}$$

Training set details

Each classification problem has its own training set. For the classification problem of Fig. 1, the networks are trained on a training set D_t of size $|D_t| = 27853$ that is artificially balanced 200-to-1 I-to-C. Classification problem (i) has a class balanced training set size of $|D_t| = 793200$. Problem (ii) has a training set size of $|D_t| = 501850$. For the classification problems (i) and (ii), the class is determined through the total number of modes $M(n)$ as described in the subsection *Numerical evidence for strip mode rules*. For the metamaterial M1 of Fig. 1, we determine the class through the rules as described in the section *Floppy and frustrated structures*.

Since there is a strong class-imbalance in the design space, for the network to learn to distinguish between class I and C, the training set is class-balanced. If the training set is not class-balanced, the networks tend to learn to always predict the majority class. The training set is class-balanced using random undersampling of the class I designs. For problem (i), with the strongest class-imbalance, the number of class C designs is artificially increased using translation and rotation of class C designs. We then use stratified cross-validation over 10 folds, thus for each fold 90% is used for training and 10% for validation. The division of the set changes from fold to fold. To pick the best performing networks, we use performance measures measured over the validation set.

To show that our findings are robust to changes in unit cell size, we also train CNNs on classification problem (i) for different $k \times k$ unit cell sizes. The size of the training set D_t for each unit cell size k is shown in Tab. A1. Increasing the unit cell size increases the rarity of C and the size of the design space. This leads a more strongly undersampled C-I boundary as we will show in the next section.

Sparsity of the training set

To illustrate how sparse the training set is for classification problem (i), we divide the number of training unit cells per class, $|D_t(\mathrm{Class})|$ over the estimated total number of $k \times k$ unit cells of that class, $|\Omega_D(\mathrm{Class})|$. We estimate this number for class C through multiplying the volume fraction of class C β in a uniformly generated set of unit cells with the total number of possible unit cells $|\Omega_D| = 4^{k^2}$: $|\Omega_D(\mathrm{C})| \approx \beta|\Omega_D|$. Likewise, we determine the ratio for class I. The resulting ratio for class C and I is shown in Fig. A6(a). Clearly, for increasing unit cell size k, the class sparsity in the training set increases exponentially. Consequently, the neural networks get relatively fewer unit cells to learn the design rules bisecting the design space for increasing unit cell size.

Table A1. Details of the hyperparameter grid search.

k	size of D_t	size of D_test
3	31180	39321
4	397914	150000
5	793200	149980
6	1620584	150000
7	292432	600000
8	1619240	144000

Moreover, the training set unit cells of different class are, on average, farther removed from one another for increasing unit cell size k. The distance between two unit cells $|\Delta X|$ is defined as the number of building blocks with a different orientation compared to their corresponding building block at the same spatial location in the other unit cell. So two $k \times k$ unit cells can at most be k^2 building blocks removed from one another, if every single building block has a different orientation compared to its corresponding building block at the same spatial location in the other unit cell. Note that we only consider *different* orientations in this definition, we do not define an additional notion of distance between orientations of building blocks.

By measuring the distance in number of different building block orientations $|\Delta X|$ between every class C to every class I unit cell, we obtain the probability density function of distance in number of different building blocks between two unit cells of different class in the training set, see Fig. A6(b). Consequently, if k increases, the networks are shown fewer examples of unit cells similar to each other, but of different class. Thus the boundary between C and I is undersampled in the training set, with few I designs close to the boundary.

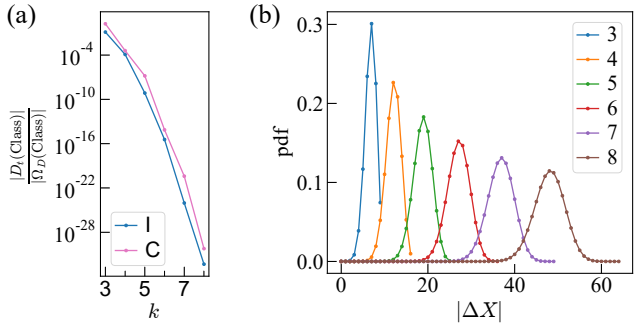

Figure A6. Training set details for classification problem (i) of metamaterial M2. (a) Fraction of the total unit cells of class C that are in the training set. (b) Average absolute distance $|\Delta X|$ in number of building blocks between class C and class I unit cells in the training set.

CNN hyperparameter grid search details

To see how convolutional neural network (CNN) size impacts classification performance, a hyperparameter grid search is performed. We focus on classification problem (i), which features a shallow CNN with a single convolution layer and single hidden layer as described in section *CNN architecture details*. This search varied three hyperparameters: the number of filters n_f, the number of hidden neurons n_h, and the learning rate η. The number of filters n_f runs from 2 to 20 in steps of 2, the number of hidden neurons n_h first runs from 2 to 20 in steps of 2, then from 20 to 100 in steps of 10. The learning rate ranges from $\eta \in 0.0001, 0.001, 0.002, 0.003, 0.004, 0.005$. For each possible hyperparameter combination, a 10-fold stratified cross validation is performed on a class-balanced training set. Early stopping using the validation loss is used to prevent overfitting.

To create the results of Fig. 4, n_f has been fixed to 20 since most of the performance increase seems to come from the number of hidden neurons n_h after reaching a certain treshhold for n_f as we will show in section *Assessing the performances of CNNs*. The best η is picked by selecting the networks with the highest fold-averaged accuracy over the validation set.

Assessing the performances of CNNs

In this section, we describe in detail how we assess the performance of our trained convolutional neural networks (CNNs). We first quantify performance over the test set, then we define our sensitivity measure. Finally, we apply this sensitivity measure to the CNNs.

Test set results

After training the CNNs on the training sets, we test their performance over the test set. The test set consists of unit cells the networks have not seen during training, and it is not class-balanced. Instead, it is highly class-imbalanced, since the set is obtained from uniformly sampling the design space. In this way, the performance of the network to new, uniformly generated designs is fairly assessed.

For the classification problem of metamaterial M1, the test set D_{test} has size $|D_{\text{test}}| = 4915$. Classification problem (i) for metamaterial M2 has test set size $|D_{\text{test}}| = 149982$. Problem (ii) for M2 has test set size $|D_{\text{test}}| = 149980$.

Precisely because the test set is imbalanced, standard performance measures, such as the accuracy, may not be good indicators of the actual performance of the network. There is a wide plethora of measures to choose from [9]. To give a fair assessment of the performance, we show the confusion matrices over the test sets for the trained networks with the lowest loss over the validation set in Tab. 1.

Varying the unit cell size

To see how the size of the unit cell impacts network performance, we performed a hyperparameter grid search as described in section *CNN hyperparameter grid search details* for $k \times k$ unit cells ranging from $3 \leq k \leq 8$. We focus on classification problem (i). The size of the test set D_{test} is shown in Tab. A1.

To quantify the performance of our networks in a single measure, we use the Balanced Accuracy:

$$\text{BA} = \left\langle \frac{1}{2}\left(\frac{V_{\text{TC}}}{V_{\text{TC}}+V_{\text{FI}}} + \frac{V_{\text{TI}}}{V_{\text{TI}}+V_{\text{FC}}}\right)\right\rangle \tag{A10}$$

$$= \left\langle \frac{1}{2}\left(\text{TCR} + \text{TIR}\right)\right\rangle, \tag{A11}$$

where V_{TC}, V_{TI}, V_{FC}, and V_{FI} are the volumes of the subspaces true class C TC, true class I TI, false class C FC, and false class I FI (Fig. 1(c, d)). We do not consider other commonly used performance measures for class-imbalanced classification, such as the F_1 score, since they are sensitive to the class-balance.

The BA can be understood as the arithmetic mean between the true class C rate TCR (sensitivity), and true class I rate TIR (specificity). As such, it considers the performance over all class C designs and all class I designs separately, giving them equal weight in the final score. Class-imbalance therefore has no impact on this score.

Despite the complexity of the classification problem, we find that, for sufficiently large n_f and n_h, the balanced accuracy BA approaches its maximum value 1 for every considered unit cell size k (Fig. A7(a)). Strikingly, the number of filters n_f required to achieve large BA does not vary with k. This is most likely because the plaquettes encode a finite amount of information—there are only 16 unique 2×2 plaquettes. This does not change with unit cell size k, thus the required number of filters n_f is invariant to the unit cell size. The number of required hidden neurons n_h increases with k, but not dramatically, despite the combinatorial explosion of the design space. To interpret this result, we note that a high BA corresponds to correctly classifying most class C unit cells as class C, and most class I unit cells as class I. Hence, sufficiently large networks yield decision boundaries such that most needles are enclosed and most hay is outside (Fig. 1(c, d)). However, whether this decision boundary coarsely (Fig. 1(c)) or finely (Fig. 1(d)) approximates the structure close to the needles cannot be deducted from a coarse measure such as the BA over the test set.

The usage of BA to show trends between neural network performance and hyperparameters is warranted,

since no significant difference between the true class C rate TCR and true class I rate TIR appears to exist, see Fig. A7. Evidently TCR and TIR depend similarly on the number of filters n_f and number of hidden neurons n_h. This is to be expected, since the networks are trained on a class-balanced training set.

The effect of class-imbalance on CNN performance can be further illustrated through constructing the confusion matrices (Fig. A8(b)). Though all CNNs show high true C and I rates, the sheer number of falsely classified C unit cells can overtake the number of correctly classified C unit cells if the class-imbalance is sufficiently strong, as for the 7×7 unit cells.

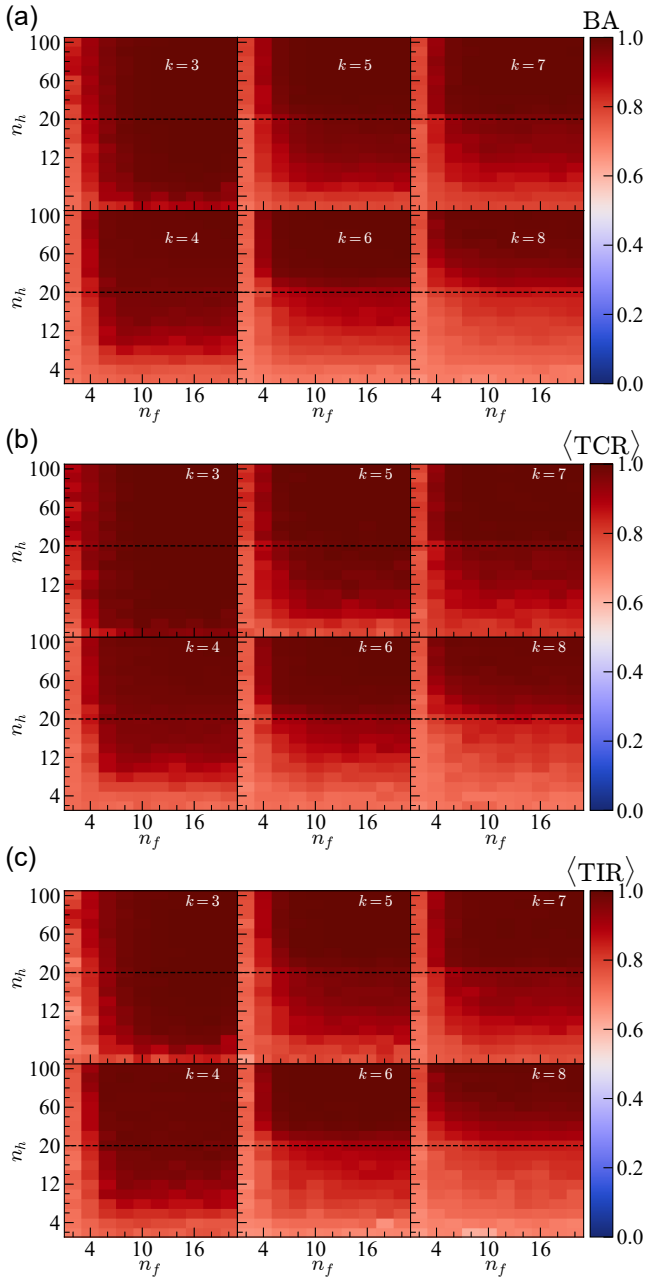

Figure A7. (a) Heatmaps of the fold-averaged balanced accuracy BA for CNNs with n_f filters and n_h hidden neurons trained on $k \times k$ unit cells indicated on top of each heatmap. (b) Heatmaps of the fold-averaged true class C rate ⟨TCR⟩. (c) Heatmaps of the fold-averaged true class I rate ⟨TIR⟩.

Increasing the size of the training set

To illustrate how the size of the training set D_t influences the performance over the test set, we compare CNNs trained on two training sets of different size consisting of 7×7 unit cells—the unit cell size with the strongest class-imbalance. We use the fold-averaged balanced accuracy BA to quantify the performance. The training sets are obtained from 1M and 2M uniformly sampled unit cells respectively, and the number of class C unit cells is artificially increased using translation and rotation to create class-balanced training sets. The best BA is more than a factor 2 smaller for CNNs trained on the larger training set, compared to the smaller training set (Fig. A9). Thus, lack of performance due to a strong data-imbalance can be improved through increasing the number of training samples.

Random walk near the class boundary

To better understand the complexity of the classification problem, we probe the design space near test set unit cell designs. Starting from a test set design X_0 with true class C, we rotate a randomly selected unit cell to create a new unit cell design X_1. We do this iteratively up to a given number of steps s to create a chain of designs. For each generated design, we assess the new true class

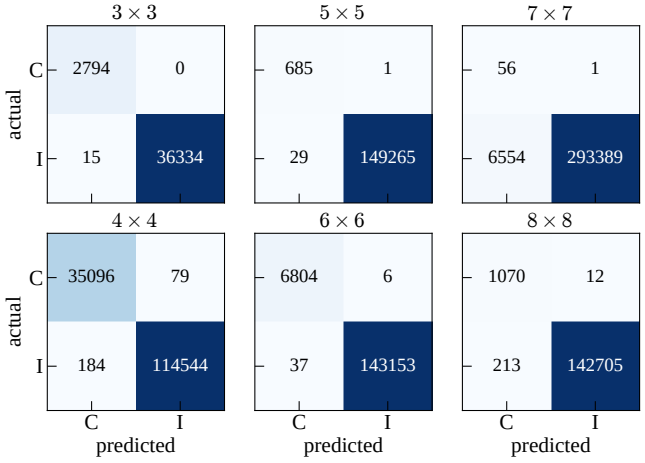

Figure A8. Confusion matrices over the test set for trained CNNs with the highest accuracy over the class-balanced validation set. The $k \times k$ unit cell size is indicated on top of each matrix.

using the design rules for classification (i) and through calculating $M(n)$ for $n \in \{3, 4\}$ for classification (ii).

For each unit cell size k, we take $s = k^2$ steps in design space. The probability to transition from an initial 5×5 design X_0 of class C to another design X_s of class C as a function of s random walk steps in design space $p_{\mathrm{C}\to\mathrm{C}}(s)$, is shown in Fig. 3(b, c) for classification problems (i) and (ii).

We repeat the random walks for other $k \times k$ unit cells for problem (i). A clear difference between the different unit cell sizes is visible. Both the rate at which the probability decreases initially, and the value to which it saturates differs per unit cell size (Fig. A10).

For even unit cell size, the dominant strip mode width is $W = 1$ (Fig. A3) and each class C design is most likely to just have a single strip mode. Thus, the probability to transition from C to I relies on the probability to rotate a unit cell inside the strip of the strip mode, which is $1/k$, so $\alpha_t \approx 1/k$. For odd unit cell sizes, the dominant strip mode width is $W = 2$, such that $\alpha_t \approx 2/k$.

To understand the asymptotic behavior, we note that for large s the unit cells are uncorrelated to their original designs. Thus, the set of unit cells are akin to a uniformly sampled set of unit cells. Consequently, the probability to transition from C to C for large s is approximately equal to the true class C volume fraction β.

Random walk near the decision boundary

In addition to the true class, we can assess the predicted class by a given network for each unit cell in the random walk. This allows us to probe the decision boundary, which is the boundary between unit cells that a given network will classify as C and those it will classify as I. By comparing the transition probabilities for given networks to the true transition probability we get an indication of how close the decision boundary is to the true class boundary.

To quantitatively compare the true class boundary with the decision boundaries, we fit the measured transition probability for each network to Eq. (1) of the Main Text with $\bar{\alpha}$ as fitting parameter. We start from designs with true and predicted class C, and track the predicted class for the random walk designs. We set the asymptotic value to the predicted class C volume fraction $\bar{\beta}$ (Fig. A11(a)) for each network. From this we obtain a 10-fold averaged estimate of $\bar{\alpha}$.

Additionally, we do this for varying unit cell size k for classification problem (i) using the hyper parameter grid search networks. We use CNNs with fixed number of filters $n_f = 20$ and varying number of hidden neurons n_h. We select the networks with the best-performing learning rate η over the validation set, and obtain a 10-fold averaged estimate of $\bar{\alpha}$ for each n_h (Fig. A11(b)).

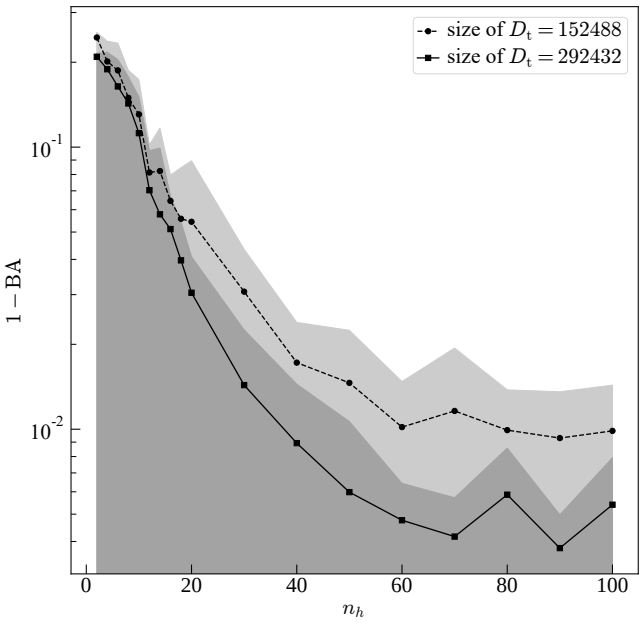

Figure A9. Balanced accuracy BA for CNNs with $n_f = 20$ trained on a smaller training set (circles) and larger training set (squares). The size of the training set is indicated in the legend.

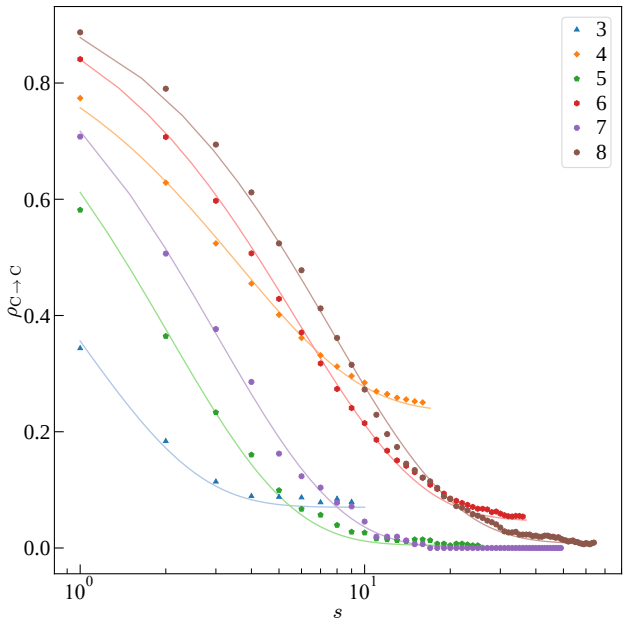

Figure A10. Probability $\rho_{\mathrm{C}\to\mathrm{C}}$ (polygons) to transition from initial design X_0 of class C to another design X_s of class C as a function of s random walk steps in design space starting from the initial design. The legend indicates the polygon and color for each unit cell size k. The continuous lines are obtained from a least-squares fit using Eq. (1) of the Main Text.

Small networks tend to overestimate the class C dimensionality α (Fig. A11(b). Larger networks tend to approach the true α for increasing number of hidden neurons n_h. For large data-imbalance, as is the case for $k = 7$ and $k = 8$, even the larger networks overestimate α. This is not a fundamental limitation, and can most likely be improved by increasing the size of the training set, see section *Increasing the size of the training set*. We conjecture that this is due to the higher combinatorial complexity of the C subspace for larger unit cells, which requires a larger number of training samples to adequately learn the relevant features describing the subspace. The trend shown in Fig. 4(c) holds across all unit cell sizes (Fig. A11(c)).

Computational time analysis

In this section we discuss the computational time it takes to classify a $k \times k$ unit cell design by calculating the number of zero modes $M(n)$ for $n \in \{2, 3, 4\}$ using rank-revealing QR (rrQR) decomposition. The first algorithm takes as input a unit cell design, creates rigidity matrices R for each n, and calculates the dimension of the kernel for each matrix using rrQR decomposition. The classification then follows from the determination of a and b in $M(n) = an + b$ as described in the main text.

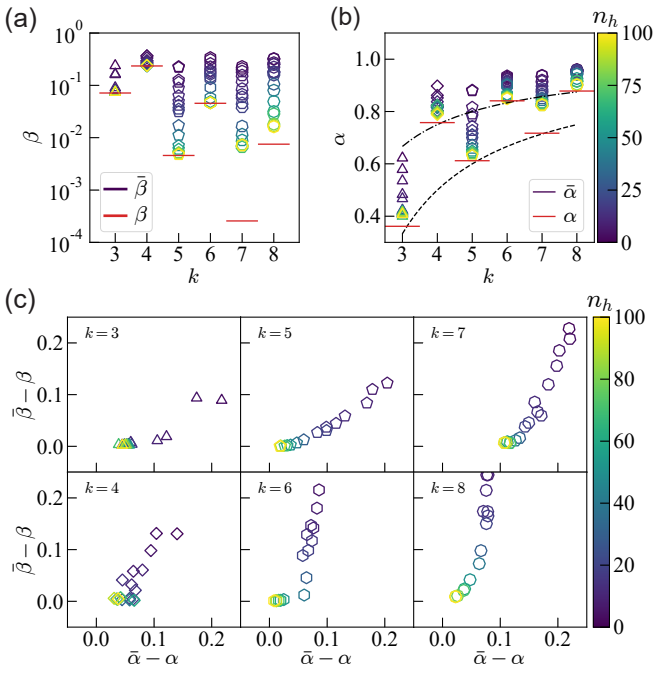

Figure A11. (a) Classification problem (i) Class C volume fraction β (red) as a function of unit cell size k. The predicted class C volume fraction $\bar{\beta}(n_h)$ (for $n_f = 20$) approaches β for increasing number of hidden neurons n_h (colorbar). (b) True dimensionality α (red) and predicted dimensionality $\bar{\alpha}(n_h)$ (colorbar) obtained through least-squares fits to data as in Fig. 3(b) for all k. The estimated α for both odd (dashed line) and even (dashdotted line) k agree well with α. (c) Scatter plots of class volume fractions $\bar{\beta}(n_h) - \beta$ versus dimensionality $\bar{\alpha}(n_h) - \alpha$ shows that the latter asymptotes later than the former (n_h indicated by a colorbar, and unit cell size k indicated on top of each graph)

We contrast this brute-force calculation of the class with a trained neural networks time to compute the classification. We consider a shallow CNN with a single convolution layer of $n_f = 20$ filters, a single hidden layer of $n_h = 100$ hidden neurons and an output layer of 2 neurons. The network takes as input a $k \times k$ unit cell design in the pixel representation (with padding) and outputs the class. The number of parameters of these CNNs grows with k, see eq.(A9). We focus on networks trained on classification problem (i).

The brute-force calculation scales nearly cubically with input size k^2, while the neural network's computational time remains constant with unit cell size k. This is due to computational overhead—the number of operations for a single forward run of our CNN scales linearly with k^2, but can run in parallel on GPU hardware. This highlights the advantage of using a neural network for classification: it allows for much quicker classification of new designs. In addition, the neural network is able to classify designs in parallel extremely quickly: increasing the number of unit cells to classify from 1 to 1000 only increased the computational time by a factor ≈ 1.33.

Please note that this analysis does not include the time to train such neural networks, nor the time it takes to simulate a large enough dataset to train them. Clearly there is a balance, where one has to weigh the time it takes to compute a sufficiently large dataset versus the number of samples that they would like to have classified. For classification problems (i) and (ii) it did not take an unreasonable time to create large enough datasets, yet brute-forcing the entire design space would take too much computational time. Our training sets are large enough to train networks on—of order 10^5—but are still extremely small in comparison to the total design space, such that the time gained by using a CNN to classify allows for exploring a much larger portion of the design space as generating random designs is computationally cheap.

* r.vanmastrigt@uva.nl

[1] C. Coulais, E. Teomy, K. De Reus, Y. Shokef, and M. Van Hecke, Combinatorial design of textured mechanical metamaterials, Nature **535**, 529 (2016).

[2] J. C. Maxwell, L. on the calculation of the equilibrium and stiffness of frames, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **27**, 294 (1864).

[3] J. N. Grima, A. Alderson, and K. Evans, Auxetic be-

haviour from rotating rigid units, Physica Status Solidi (b) **242**, 561 (2005).
[4] C. Coulais, C. Kettenis, and M. van Hecke, A characteristic length scale causes anomalous size effects and boundary programmability in mechanical metamaterials, Nature Physics **14**, 40 (2018).
[5] A. Bossart, D. M. Dykstra, J. van der Laan, and C. Coulais, Oligomodal metamaterials with multifunctional mechanics, Proceedings of the National Academy of Sciences **118**, 21 (2021).
[6] There is a small and exponentially decreasing portion of unit cells that requires to calculate $M(n)$ with $n = 5$ and 6 to determine whether they belong to class I or C. We leave these out of consideration in the training data to save computational time.
[7] See `https://uva-hva.gitlab.host/published-projects/CombiMetaMaterial` for code to check the rules.
[8] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
[9] M. Hossin and M. N. Sulaiman, A review on evaluation metrics for data classification evaluations, International Journal of Data Mining & Knowledge Management Process **5**, 1 (2015).

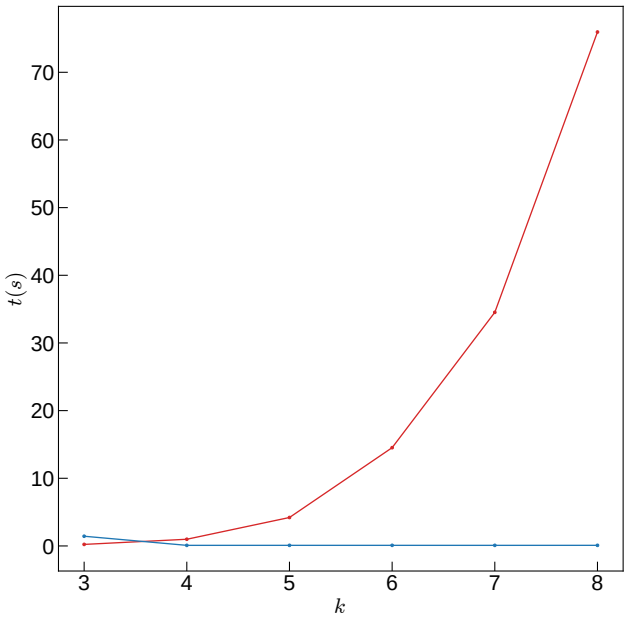

Figure A12. Computation time t measured in seconds s to classify $k \times k$ unit cells by total number of modes $M(n)$ (red) versus using a trained convolutional neural network (blue).