Identifying Discrete Breathers using Convolutional Neural Networks

Thesis report submitted in partial fulfillment of the requirements for the degree of

Bachelors of Science in Physics

by

Thomas Dogkas,

Under the supervision of

Prof. Georgios Tsironis

Department of Physics

University of Crete

24 October 2022

Contents

Appendices (Code)

Acknowledgments

This Thesis was a great opportunity for me to improve and discover what career path I want to follow. Thus, I would like to express my deepest gratitude to Mr. Tsironis and his college Mr. George Barbaris for giving me the opportunity to work on the project and guiding from the beginning to end with useful advice. I also want to thank Maria Eleftheriou for providing me with insight on the topic due to her past experience working on breathers and also data that were proven to be very useful for the final evaluation of the project. It was a challenging task for me and without any of their help I wouldn't be able to finish this thisis. So, for all that I am truly grateful for their help. Last but all least I would also like to all of my friends and family that helped by standing on my side through thick and thin in all those years as a university student.

Heraklion, Crete
24 October 2022

Thomas Dogkas

Abstract

Discrete Breathers (DB) are the solutions of a non-linear partial differential equation in discrete lattice that are time-periodic and space localized. By constructing images of DBs and phonons with 3 different non-linear potentials we will try to use Convolutional Neural Networks to classify DBs from phonons and also to identify the potential that each image was created.

INTRODUCTION

A breather is a time-periodic and space-localized solution of a non-linear partial differential equation. A discrete breather is a similar localized periodic solution in a discrete lattice. Discrete Breather (DB) or Intrinsic Localized Modes (ILMs) first conception was made by Takeno and Sievers [3-5] in the 80s and ever since it has been a topic of research due to their long life-spam with almost no energy loss [4], studying the dynamical properties of DBs and their implementations. Some of the research around DB include studies of non-linear symmetric and magnetic materials [1,2], non-linear models in thermal denaturation of DNA [6], the Josephson junctions [7], in biological systems such as the molecules of Myoglobin [8], the presence of BDs in photonic crystals [9], anti-ferromagnets [10] and in hydrocarbon and carbon [11].

In this thesis discrete breathers and phonons in 1D lattice space were constructed using 3 different non-linear potentials and small couplings in order to create a small dataset with2D images of breathers and phonons. The purpose of this was to try by using neural networks specifically Convolutional Neural Networks (CNN) to try and distinguish breathers from phonons, as well the potential based only from the images we produce and compare the results with those of supervised machine learning models.

The thesis consists of 2 parts. Firstly, the creation of breathers and phonons and then the construction of our CNN for the classification. In the first part we start by defining the phonon spectrum for each potential and coupling then by using the Runge Kuta method we define the frequency for different Amplitudes and constructing the breather or phonons depending if the frequency belongs to phonons spectrum or not. In case of breathers the stability was check with Floquet theory and by visualizing time evolution of the solution for few periods. Finally, all the solution were used again as initial conditions in order to produce the 2D images of the lattice as it evolves in time. In the second part we feed the CNN with the data once for trying to classify breathers from phonons and once for trying to identify the potential. The final dataset consists of 459 sample 10 of which were constructed and given from Maria Eleftheriou for mixing up the dataset with solution not constructed in the same way.

SECTION 1: CONSTRUCTION OF THE SAMPLES

Mathematical background of discreate breathers

Studying the effects of nonlinearity and discreteness on the spatial localization of discreate breathers, we start by looking at the one-dimensional chain of interacting oscillators with following Hamiltonian:

In this equation we have the non-linear potential V at position x_n and W potential of the one-dimensional chain of interacting oscillators at the same position. Also, the integer n is the position of a possibly infinite chain in one dimensional lattice and p_n the momentum. The equations of motion of the above Hamiltonian read as follows:

$$H = \sum_N \frac{{p_n}^2}{2} + V(x_n) + W(x_n) \quad (1)$$

If we linearize the equations, we obtain around the classical ground state $x_n=p_n=0$ by having V'(0) =W'(0) and V''(0), W''(0)≥0 coupled differential equations with the solutions being small amplitude plane waves:

$$x_n(t) \sim e^{i(\omega_b t - qn)} \quad (2)$$

$${\omega_b}^2 = V''(0) + 4W''(0)\sin\left(\frac{q}{2}\right) \quad (3)$$

The equation (3) gives the phonon frequency spectrum. In order to have breather solutions we need to solve the equations of motion with frequencies outside that spectrum.

We define a potential V(x) and the one-dimensional chain of interacting oscillator potential W(x):

$$W(x) = (x_{n+1} + x_{n-1} - 2x_n) \quad (4)$$

By solving the equation of motion with zero coupling basically ignoring the oscillator potential we get a for a specific amplitude the frequency that in the that site will have depending only on potential V. Then we create a Vector $U(x_1,x_2,....,x_n,p_1,p_2,p_n)$ of N sites in lattice space and setting the in middle site with

the initial condition x_n and p_n=0 we solve the equations of motions again by slowly increasing the coupling k and solving it again with p_n=0 . Then we have

$$U_{k+\delta k} = U_k + \Delta \quad (5)$$

Due to time periodicity, we have:

$$U_k + \Delta = T(U_k + \Delta) \Leftrightarrow U_k + \Delta = T(U_k) + \partial T \, x \, \Delta \quad (6)$$

where $\partial T = M$ the tanged map. The Matrix M can be obtained by the linearization of the equations of motion with the equation:

$$\ddot{\varepsilon_n} + V''(x_n)\varepsilon_n = k(\varepsilon_{n-1} + \varepsilon_{n+1} - 2\varepsilon_n) \quad (7)$$

where ε_n is an array of the 2N x 2N identical matrix.

The Matrix can also be translated in the following equation:

$$M = I_{2N} + \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad (8)$$

Thus, minimizing the square of norm:

$$\|T(U) + M\Delta - (U + \Delta)\|^2 \quad (9)$$

We finally get the equation:

$$\delta = -(A^*A + C^*C)^{-1}(A^*, C^*)(T(U) - U) \quad (10)$$

And by deriving the δ from the solution we have our breather.

Phonon Frequencies band

In order to define the frequencies that are capable to create breathers we had to define the frequencies that we have phonons by using the equation (3). We define the upper and lower limit of the phonon band. When we have Frequencies $\Omega_b < \omega_b$, where ω_b the frequency of the of the phonon spectrum and Ω_b the frequency of the breather, the created breathers are acoustic-like while when we have $\Omega_b > \omega_b$ are optic-like. In order to define Ω_b we solve the equation of motion for coup link k=0 meaning that there is no interaction between each site on the one-dimensional lattice and observe the period of the osculation for the perpetuated site.

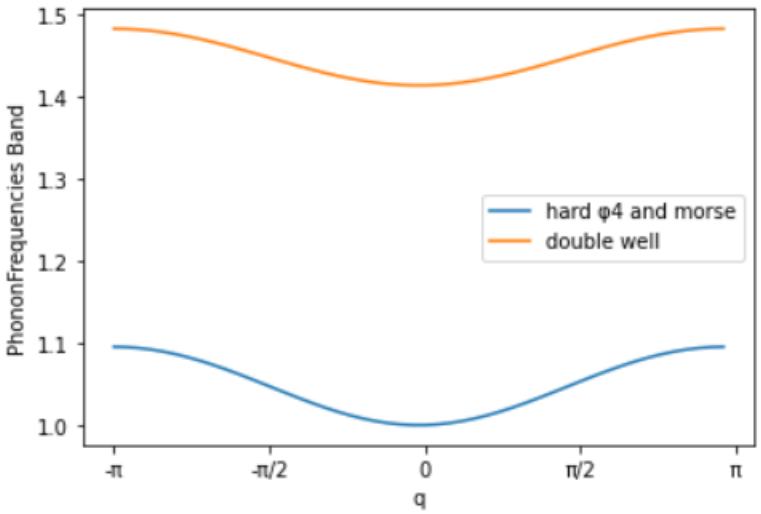

Fig 1: Phonon Frequency band with upper limit ω_b= 1.095, lower limit ω_b= 1 and coupling k=0.05 for hard ϕ4 and morse potentials and with upper limit ω_b= 1.483, lower limit ω_b= 1.414 .

Nonlinear Potentials for creating breathers

For this analysis we used 3 different non-linear potential:

- Hard φ4: $V(x) = \frac{x^2}{2} + \frac{x^4}{4}$ (11)
- Morse: $V(x) - \frac{1}{2}(1 - e^{-x})^2$ (12)
- Double Well: $V(x) = -\frac{(x-1)^2}{2} + \frac{(x-1)^4}{4}$ (13)

By finding the frequency of an oscillator in one dimensional lattice of N=40 site points and using it to find the period of the oscillation we solve the equation of motion and the linearized solution for finding the tangent map of the system through the equation (7) for that specific period. Then by increasing our coupling k with a small step dk=0.001 we looping this procedure until we have our solution when k equals to our targeted coupling.

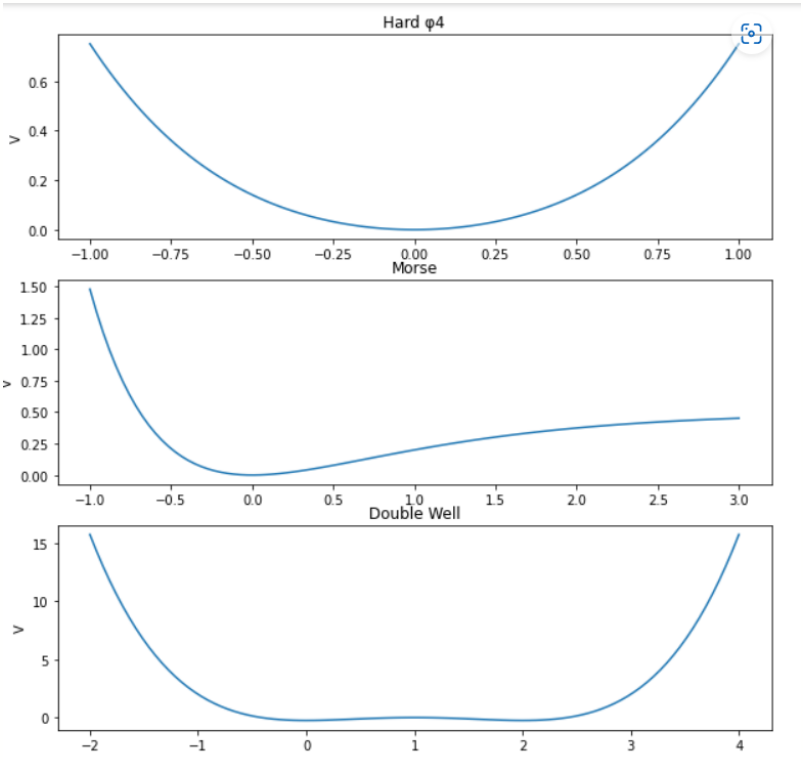

Fig 2: Visualization of the potentials

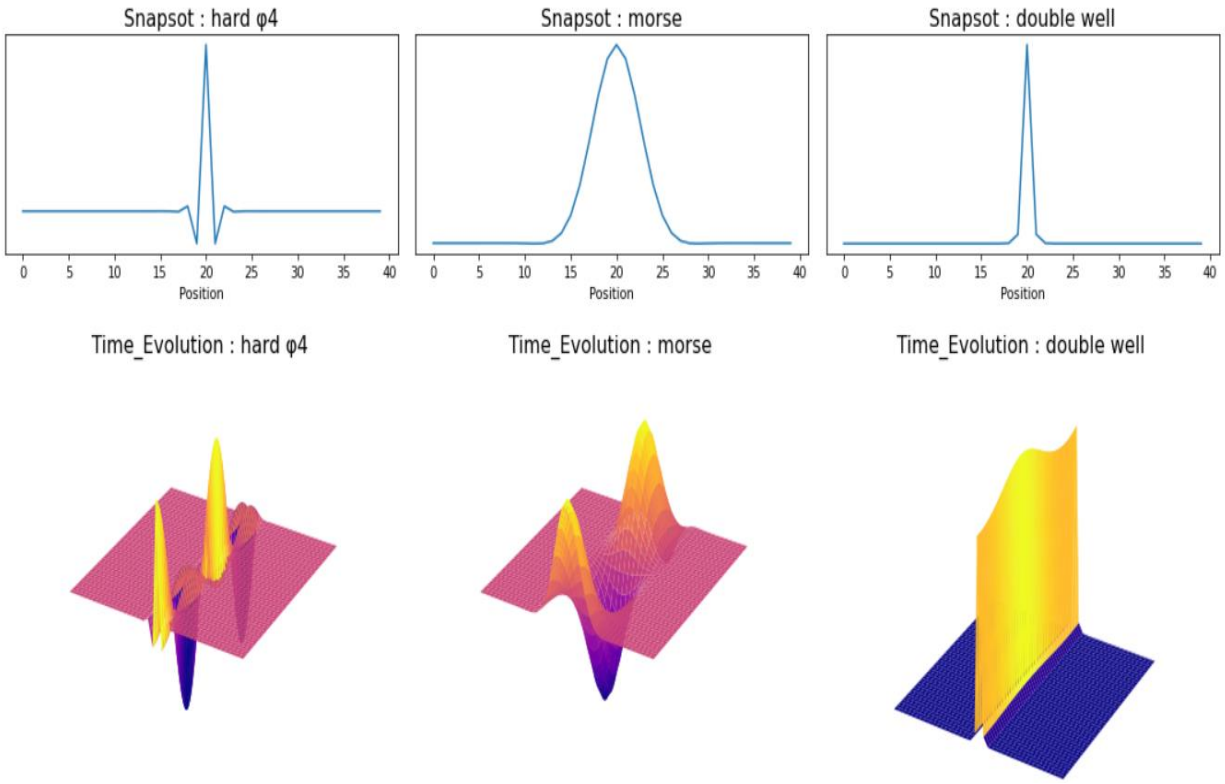

Fig 3: Breather images and their time evolution for hard ϕ4, morse and double well potentials with coupling k=0.1 and frequencies f=1.317, f=0.967 and f=0.949 respectively.

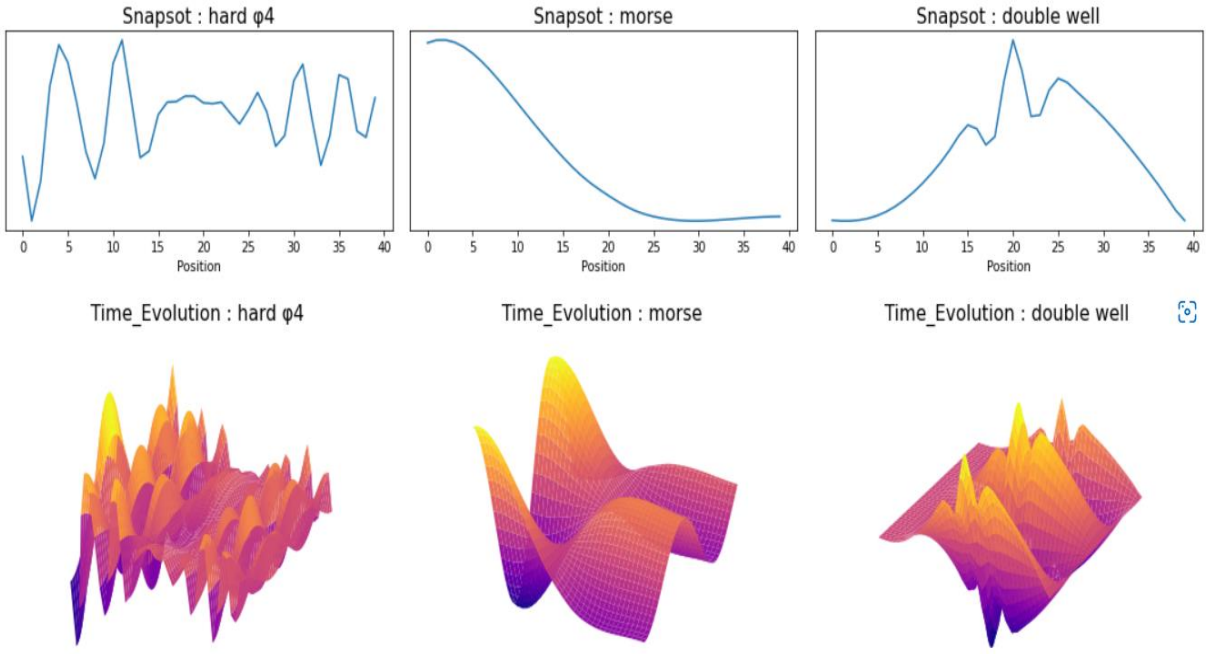

Fig 4: Phonons images and their time evolution with potentials hard ϕ4, morse and double well with coupling k=0.1 and frequencies f=1.042, f=1.001, f=1.415 respectively

Stability of desecrate breathers

For finding out if a breather is stable or not, the floquet multipliers had been used. Those can be obtained by defining the eigenvalues of the matrix M which represents the tangent map of our system of equation (8). One other criterion of desecrate breathers is their long lifespan over a long period of time.

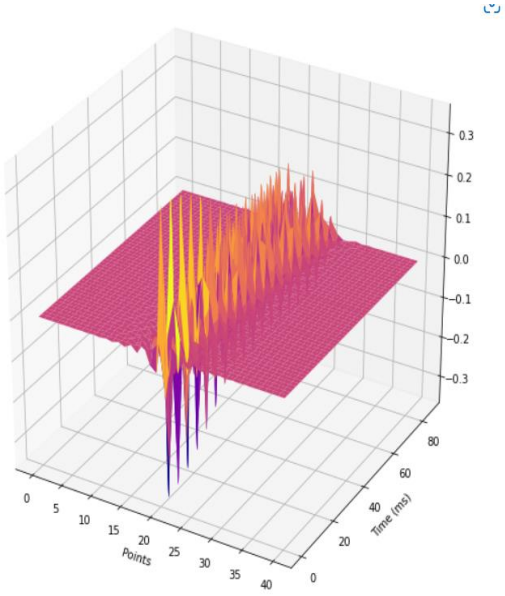

Fig 4: Unstable breather with hard ϕ4 potential frequency f=1.099 and coupling k=0.05 for 15 periods of time

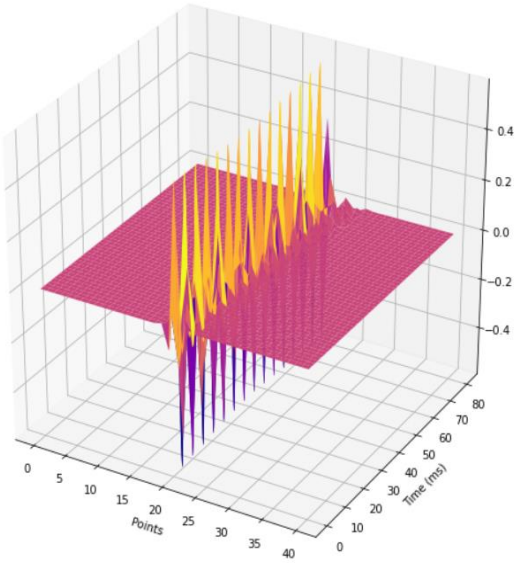

Fig 5: Stable breather with hard ϕ4 potential frequency f= 1.17 and coupling k=0.05 for 15 periods of time

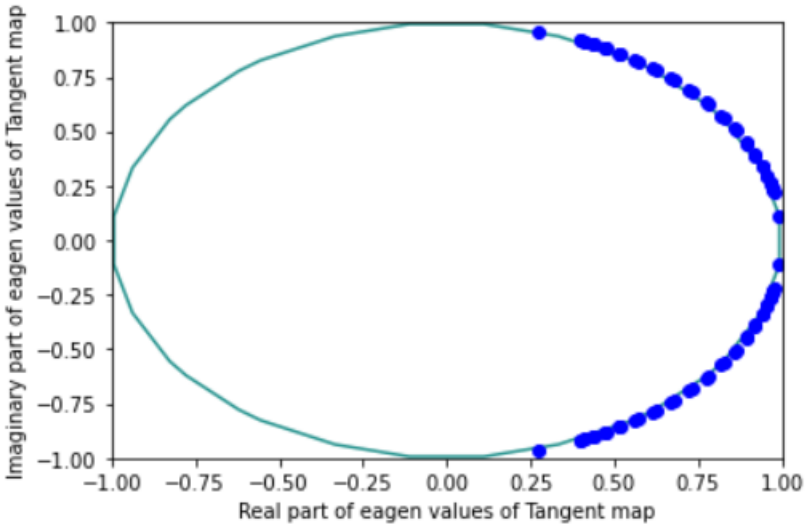

Fig 6: Eigen values of tangent map M matrix of a breather with potential hard ϕ4, frequency f=1.227 and coupling k=0.1

SECTION 2: NEURAL NETWORKS AND CNNS

Neural Networks

A neural network is a two-stage regression or classification inspired by human brain models. As an input it takes a vector X with k elements leading to an output Y with k labels. Between the input and the labels hidden layers Z^m exist with m≥1 the number of layers that fully connect the inputs through the last layer T with dimension equal to X with the outputs. Each value of the Z^m hidden layers is called a node and via an activation function σ each node of a layer connected with a node of the next one. Lastly the prediction g(T)=f(X). Each vector consists from hidden parameters called weights and each value can be denoted those weights θ as followed:

$\{a_{0,}\ldots,a_m ; m=1,\ldots M\}$ M(p+1) weights.

When the inputs feed the network the error of prediction is calculated

- Regression error R(θ) $=\sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik}+f_k(x_i))^2$ (14)
- Cross-entropy error R(θ)$=-\sum_{k=1}^{K}\sum_{i=1}^{N}y_{ik}\log(f_k(x_i))$ (15)

Then via Gradient decent a procedure called back-propagation occurs to update the data

- Gradient Decent and Back-propagation

  Assuming we have an input X with each value denoted to weights $\{a_1,\ldots a_m; m=1,\ldots M\}$ and a hidden layer Z with each value denoted to weight $\{b_1,\ldots b_k; k=K\}$:

  - $Z=\sigma(\alpha_0+\alpha_1X+\cdots+a_MX)$ (16)
  - $T=b+b_1Z+\cdots+b_KZ$ (17)

  By taking the derivatives of R with respect to a, b

- $\frac{\partial Ri}{\partial b_{km}}=-2(y_{ik}-f_k(x_i)g_k'(b_k^Tz_i)z_{mi}$ (18)

- $\frac{\partial Ri}{\partial a_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i) g'_k(b_k^T z_i) z_{mi}\, \sigma' g'_k(a_m^T x_i) x_{li}$ (19)

And those derivatives are getting iterated r+1 times we get:

- $b_{km}^{(r+1)} = b_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial Ri}{\partial b_{km}^{(r)}}$ (20)
- $a_{ml}^{(r+1)} = a_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial Ri}{\partial a_{ml}^{(r)}}$ (21)

Those equation can be translated as followed:

- $\frac{\partial Ri}{\partial b_{km}} = \delta_{im} z_{im}$ (22)
- $\frac{\partial Ri}{\partial b_{km}} = s_{im} z_{il}$ (23)

Thus, giving us the equation of back-propagation:

- $s_{im} = \sigma'(a_m^T x_i) \sum_{k=1}^{K} b_{km} \delta_{ki}$ (24)

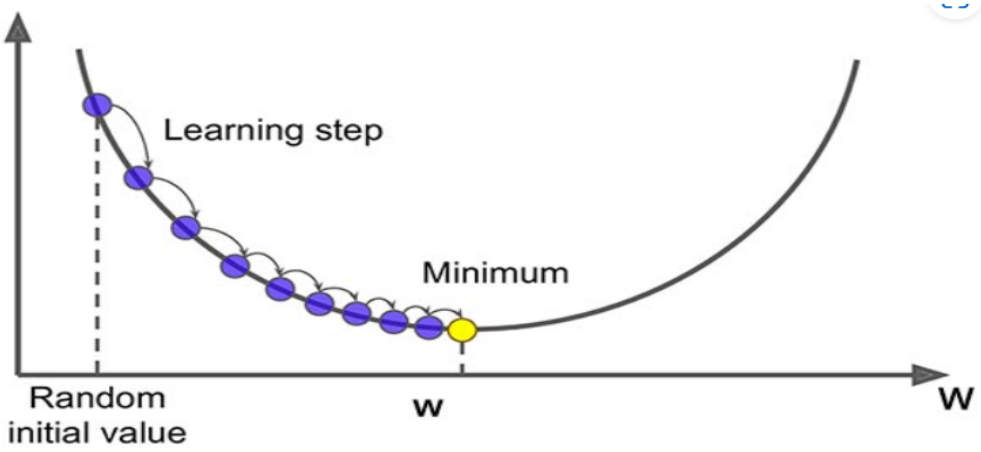

Fig 7: Visualizing gradient decent algorithm

By calculating the errors δ_{im} we update the errors s_{im} via (24) and update the values for gradient descent of equations (20), (21). This Backward pass algorithm for implementing gradient decent is called back-propagation.

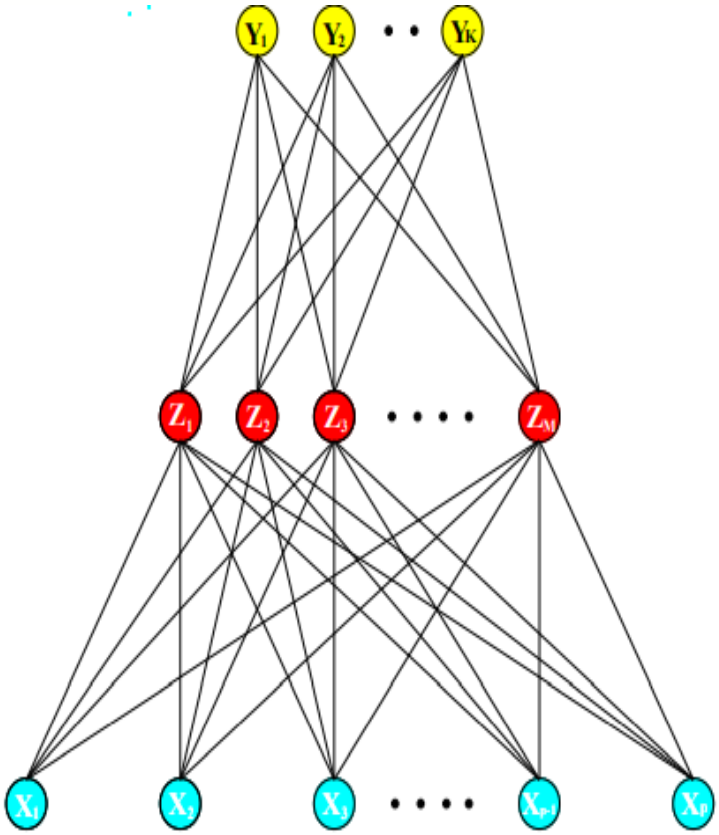

Fig 8: A visualization of a single layer neural network.

Convolutional Neural networks

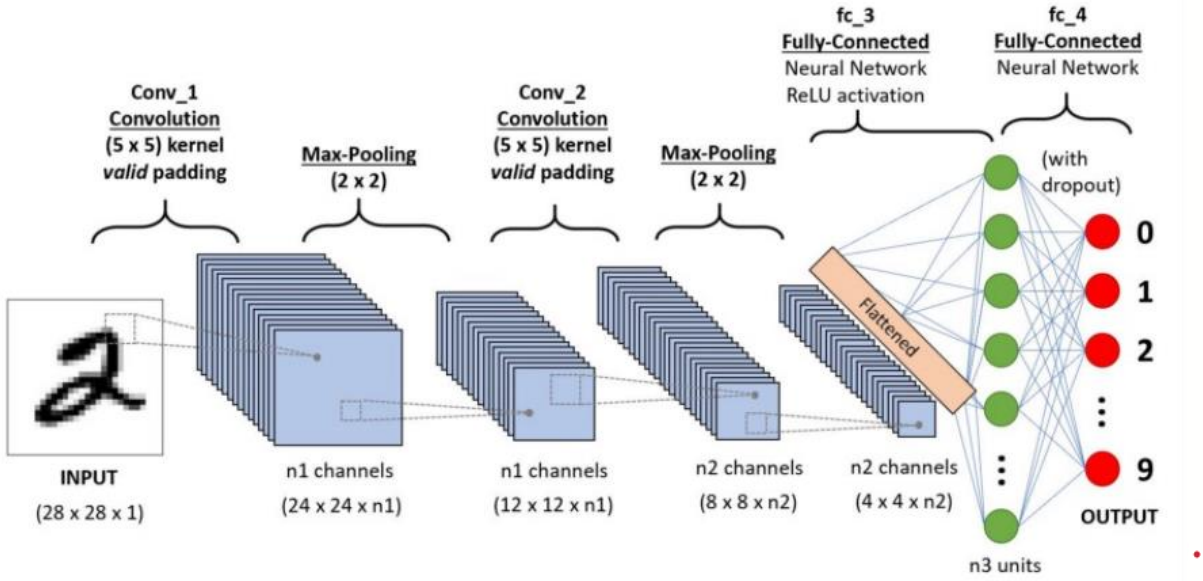

Fig 9: A CNN sequence for classifying hand writing

In order to distinguish images based on some characteristics those may have CNNs are mainly use. A CNN is a fully connected neural network that takes as input an initial image with MxN dimensions and by contacting a convolution operation produces a squeezed image of the previous layer containing all the high-level features of that image. It is possible to even more reduce the dimensionality of the image by using a pooling method by implementing a kernel of some nxn dimensions. There are 2 kinds of pooling Maxpooling and Avarage pooling. On the last step the image is flattened in order be connected Feed-forward for classify and connect the extracted features with the dataset labels.

- 2D Convolutional operation: Is an operation where an initial image is convolved through a kernel. The kernel which is a nxn matrix goes through the initial image multiplying each nxn part of that and compress that image keeping the high-levels features of those images such as edges or color intensity

  The mathematical formula is denoted as followed:

$$H_{i,j} = \sum_n \sum_n K_{i,j} X_{i+a,j+b} \quad (25)$$

Where the matrix is the convolved imaged the matrix X is the Kernel.

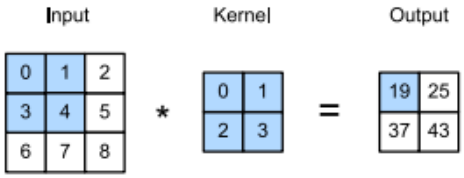

Fig 10: convolution of the Input matrix with the kernel matrix

- Max pooling: Max pooling is a dimensionality reduction in which through a kernel calculates the max value of the input matrix in the position of the kernel

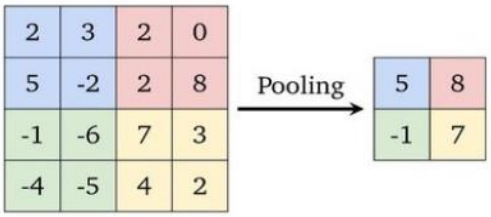

Fig 11: Max pooling of a 4x4 Matrix through a 2x2 kernel

- Average pooling: Average pooling is a dimensionality reduction in which a kernel calculates the average value of input in the position of the kernel

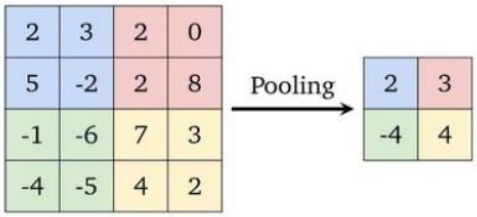

Fig 12: Average pooling of a 4x4 matrix through a 2x2 kernel

SECTION 3: TRAINING PROCESS AND DISCUSSION OF THE RESULTS

Training process

In the first step we created images of breather in order to feed our CNN. There were created 449 sample of phonons and breather created with the potentials (11), (12), (13) and the labels of their frequencies, couplings, potentials and their class as phonons or breathers from whose we keep the potential and class labels. The dataset was shuffled 10 breathers added to the dataset created from Maria Eleftheriou in order to add randomness to our training for avoiding biases (training a CNN including mainly breathers or phonons or sample created with same potential). Then we split the dataset with an 80%-20% of training and testing sets accordingly, the training set was normalized using the ImageDataGenerator package from keras.preprocessing library and the labels was categorized. For classification the images were used with one channel so the dataset shape was (N, T,1) where N, T are the 1D lattice space and the time period respectively. In the training model we used 3 2D convolution layers and 2 Max pooling layers in between. In the training process the training set was further split again with 80%-20% ratio for training end validation set. For calculating the errors of the training, the cross-entropy loss function was used and for optimizing the hyper parameters like learning rate the adam optimizer was used. The initial epoch in which the model would run was set to 100 with an early stopping parameter in respect to validation loss (errors of the validation set inside the training) was set with patience 5 in order to avoid overfitting and set to keep the epoch with the lowest validation loss.

Results

- **Model for distinguish breather from phonons**

  The Algorithm lasted for 18 epochs with the lowest validation loss to be at 15^{th} epoch as it is shown in the figure 13.

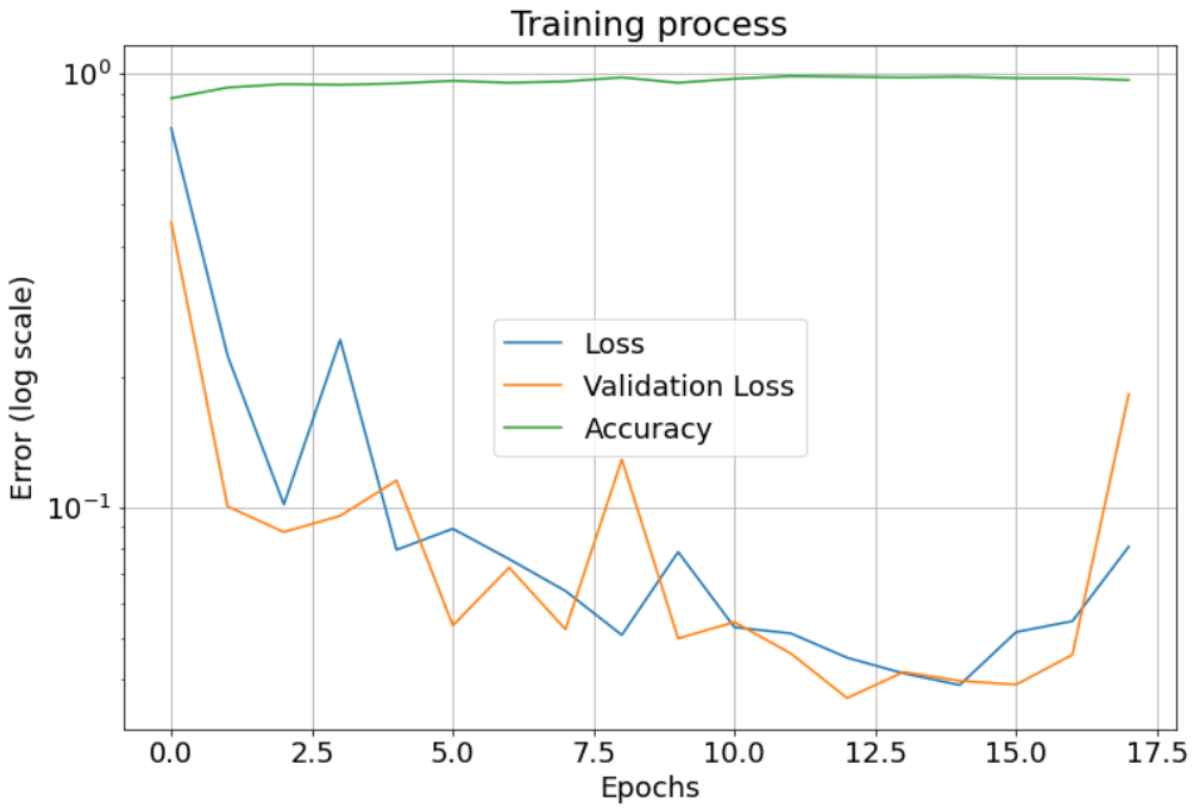

Fig 13: Plots of accuracy, total errors in log scale and errors in in validation process

Our model quickly achieves great accuracy and had a small amount of both Loss and Validation Loss around 10^{-1}.

The metric we used for evaluation of the model was f1_score which show the percentage of True Negative and True positive predictions. The result of our model was 95% and that can be seen from the confusion matrix of fig 14 where it shows that from the 92 samples used for testing only 4 were falsely predicted.

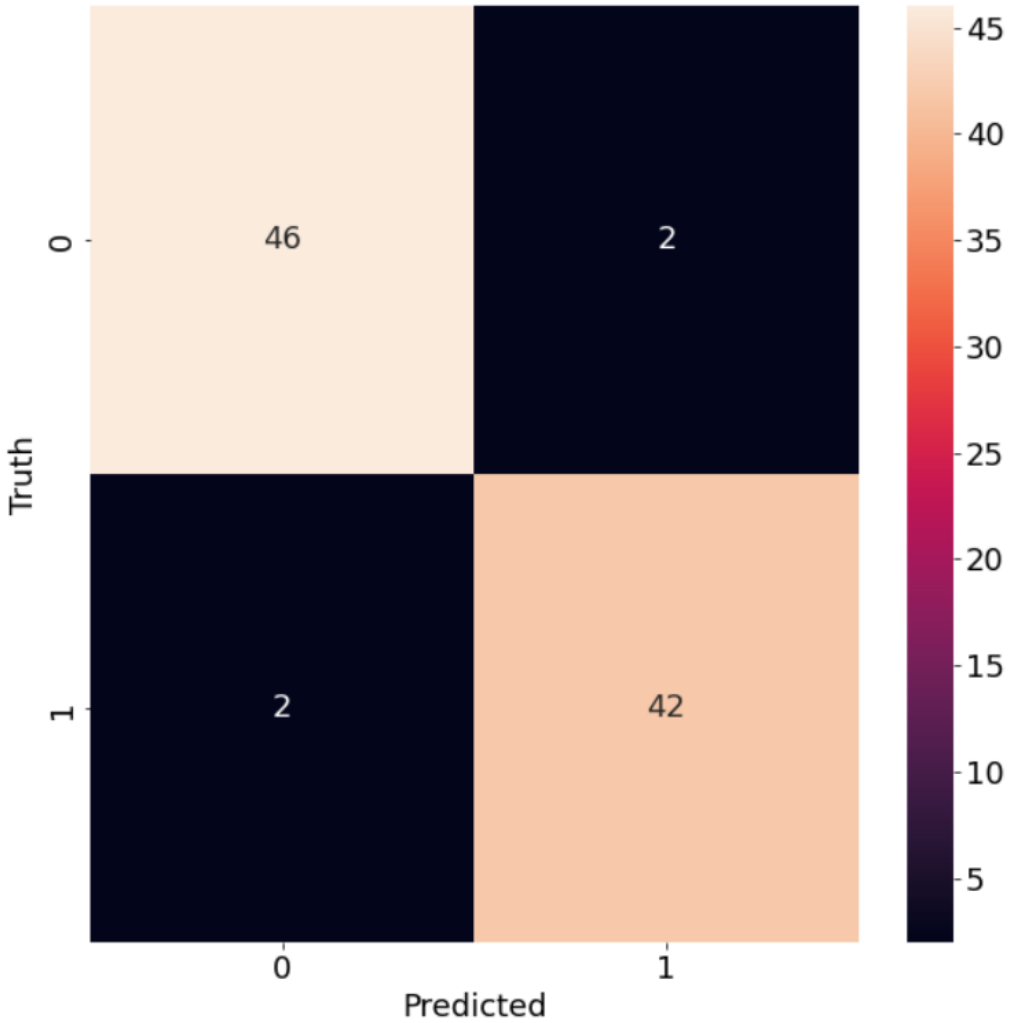

Fig 14: Confusion matrix of our results shown the true positive and true negative predictions where 1 indicates breathers and 0 indicate phonons.

- **Model for distinguish the potentials**

The Algorithm lasted for 9 epochs with the lowest validation loss to be at 4^{th} epoch as it is shown in the figure 15.

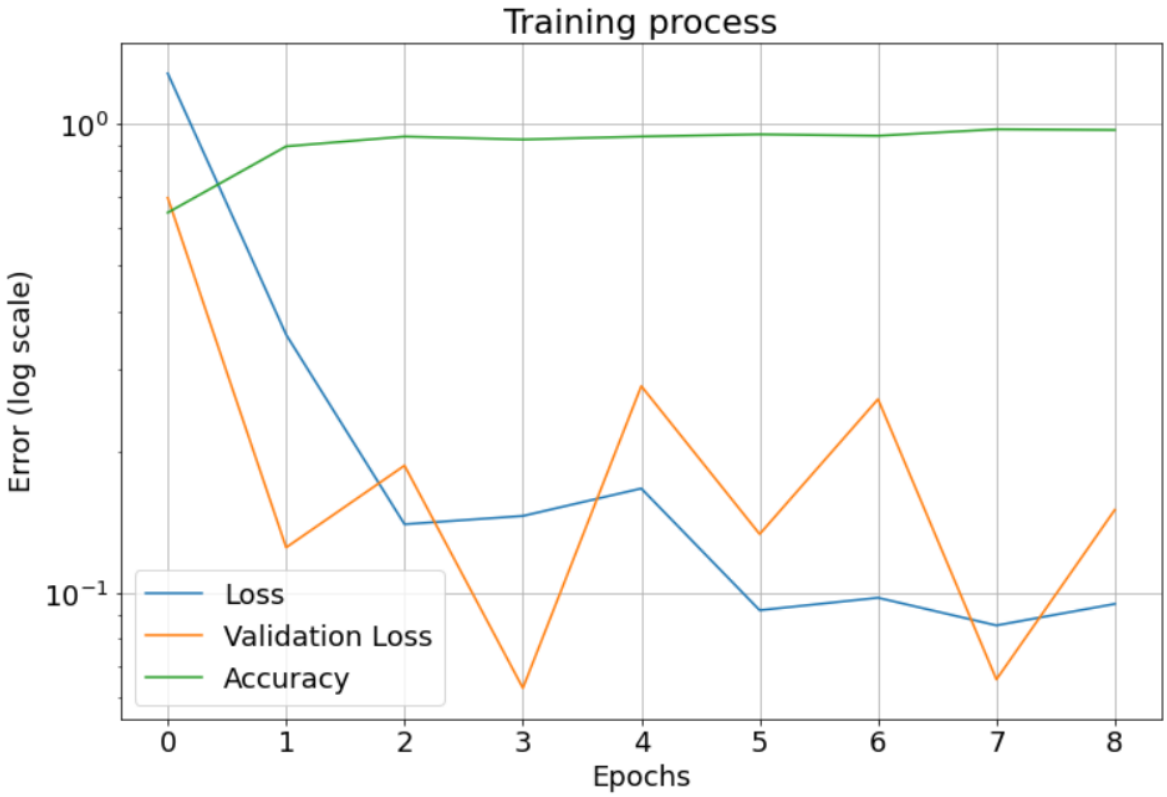

Fig 15: Plots of accuracy, total errors in log scale and errors in in validation process.

The result of our model was 93% and that can be seen from the confusion matrix of figure 16 where it shows that from the 92 samples used for testing only 6 were falsely predicted.

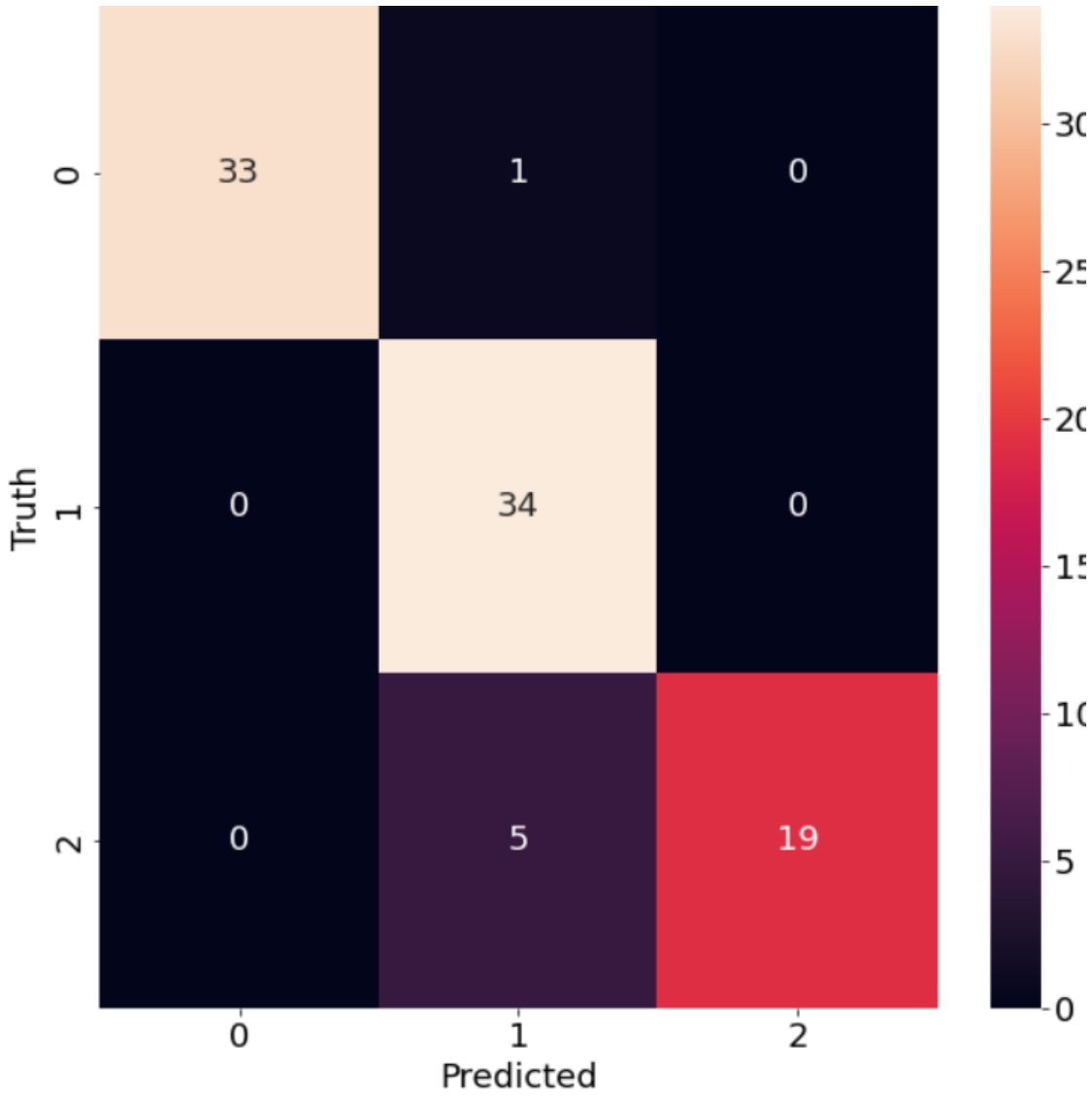

Fig 16: Confusion matrix of our results shown the true positive and true negative predictions where 0 indicates morse potential, 1 indicate double well potential and 2 indicates hard ϕ4 potential.

	Model for classifying breather and phonons	**Model for Distinguishing the potentials**
Best Epoch	12	3
Loss (error) of the model	0.0449	0.1463
Accuracy of the model	0.9829	0.9283
Validation loss (validation error)	0.0362	0.0629
Validation accuracy	0.9865	0.9865
F1 score	0.9545	0.9341

Table of the total result of the CNN model for both classifying breathes and phonons and the potentials.

Discussion

By exploiting the localization of discrete breathers in lattice space and constructing the breathers in such a way that the initial perpetuation to be in the middle of the one-dimensional chain n=N//2 where N=40 sites, images of breathers and phonons were constructed through the time evolution of the solution. Example of those images are shown on the figures 8 and 9. Since the distinction of the two images are clearly visible a model that could evaluate if an image was that of a breather or a phonon could be made. The results of our model shown great promise that a convolutional neural network can identify breathers from phonons pretty well, thought in this particular thesis there were used only breather and phonons that were constructed by the equations (11), (12) and (13) meaning that there might be breather solution constructed with different potentials that our model might not be able to classify them with such accuracy. Comparing the results of those done with supervised training methods [16], it seems that both approaches can produce very good results.

Fig 17: Image of the time evolution of a breather with double well potential, coupling k=0.1 and frequency f=1.332

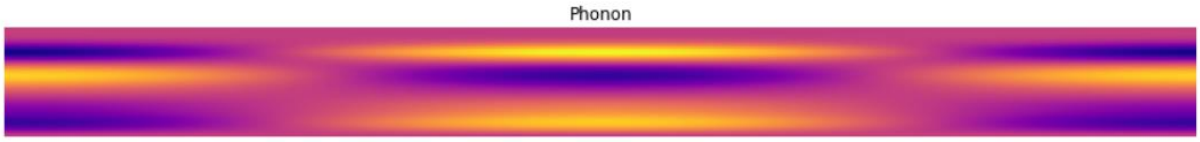

Fig 18: Image of the time evolution of a phonon with double well potential coupling k=0.1 and frequency f=1.416

For better visualization a breather and phonon image contour plots of the 2 were created as they shown in figures 19 and 20.

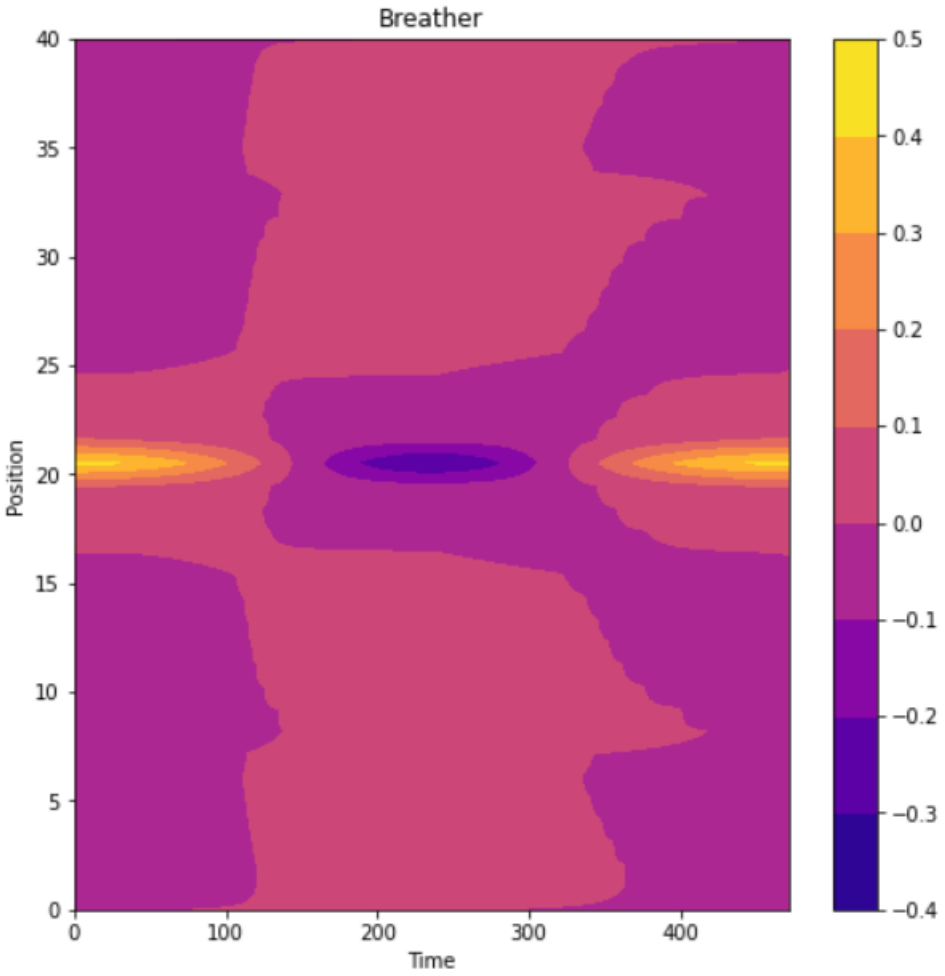

Fig 19: Contour plot of a breather with double well potential, coupling k=0.1 and frequency f=1.332

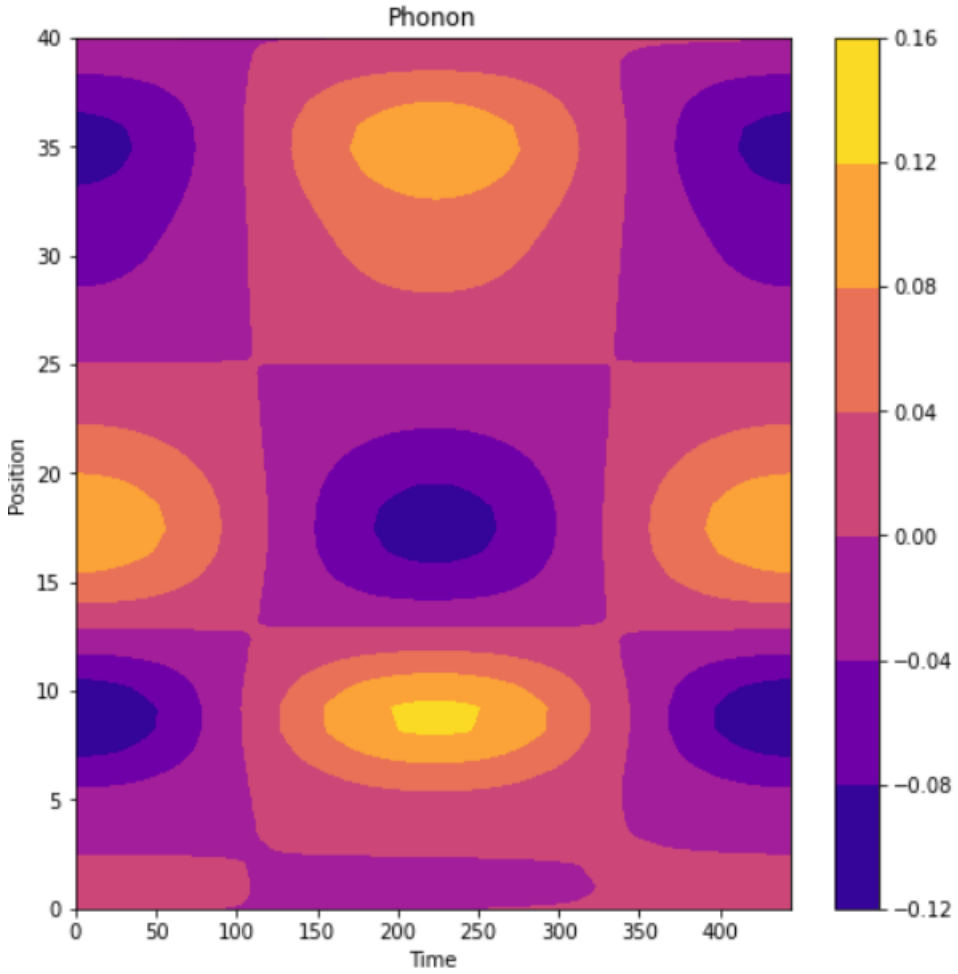

Fig 20: Contour plot of a phonon with double well potential, coupling k=0.1 and frequency f=1.416

BIBLIOGRAPHY

[1] N. Lazarides, M. Eleftheriou, G. P. Tsironis, Discrete breathers in nonlinear magnetic metamaterials, Physical review letters Vol. 95 Issue 15, 12-10-2006.

[2] N. Lazarides, G. P. Tsironis, Gain-Driven Discrete Breathers in Symmetric Nonlinear Metamaterials, Physical review letters Vol. 110 Issue 5, 30-1-2013.

[3] Dolgov, A. S., The localization of vibrations in a nonlinear crystal-structure. Fizika Tverdogo Tela 28 (6), 1641 (1986).

[4] Sievers, A. J. and Takeno, S., Intrinsic localized modes in anharmonic crystals. Physical Review Letters 61 (8), 970 (1988).

[5] Page, J. B., Asymptotic solutions for localized vibrational modes in strongly anharmonic periodic systems. Physical Review B 41 (11), 7835 (1990).

[6] Nonlinear localization in thermalised lattices: application to DNA, M. Peyrard, J.Farago Physica A 288, 199(2000)

[7] Trias, E., Mazo, J. J., and Orlando, T. P., Discrete breathers in nonlinear lattices: experimental detection in a Josephson array. Physical Review Letters 84 (4), 741 (2000).

[8] Long-lived A, mide I vibrational modes in myoglobin, A. Xie, L. van der Meer, W. Hoff, R. H. Austin, Phys. Rev. Lett. 84, 5435 (2000)

[9] Fleischer, Jason W., Segev, Mordechai, Efremidis, Nikolaos K., and Christodoulides, Demetrios N., Observation of two-dimensional discrete solitons in optically induced nonlinear photonic lattices. Nature 422 (6928), 147 (2003).

[10] Sato, M. and Sievers, A. J., Direct observation of the discrete character of intrinsic localized modes in an antiferromagnet. Nature 432 (7016), 486 (2004).

[11] J.A. Baimova, A. Korznikova,I.P. Lobzenko nd S.V. Dmitriev DISCRETE BREATHERS IN CARBON AND HYDROCARBON NANOSTRUCTURES, Rev.Adv. Mater. Sci. 42 (2015) 68-82

[12] ASTON ZHANG, ZACHARY C. LIPTON, MU LI, AND ALEXANDER J.SMOLA, Dive into Deep Learning , Amazon Science Martch 25 2022

[13] Maria Eleftheriou, Statistical Properties of Classical Non-Linear Lattice, Phd Thesis for University of Crete 2003

[14] Sergej Flach , Discreate breathers in a nutshell, NOLTA IEICE January 2012

[15] Trevor Hastie, Robert Tibshirani , Jerome Friedman, The Elements Of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer, 2009)

[16] Janis Bajars, Filips Kozirevs, Data-driven intrinsic localized mode detection and classification in one-dimensional crystal lattice model, Physics Letters A 436 (2022) 128071

APPENDICES (CODE)

Equation of motion for hard φ4, morse and double well potentials respectively

```
def ac_hard(t,xu,k):

    x, u = xu[:N], xu[N:] # Split xu vector into x and
    u.#
    dxdt = u # New position.
    dudt = np.zeros((N,1)) # New velocities.
    dudt[1:N-1] = -x[1:N-1]-x[1:N-1]**3 - 2*k*x[1:N-
    1] + k*(x[0:N-2] + x[2:N])
    dudt[0]= -x[0]-x[0]**3 - k*(x[1]-x[0])
    dudt[N-1]= -x[N-1]-x[N-1]**3 - k*(x[N-1]-x[N-2])

    return np.append(dxdt,dudt)

def ac_morse(t,xu,k):

    x, u = xu[:N], xu[N:] # Split xu vector into x and
    u. #
    dxdt = u # New position.
    dudt = np.zeros((N,1)) # New velocities.
    dudt[0]= -np.exp(-x[0])+np.exp(-2*x[0]) - k*(x[1]-
    x[0])
    dudt[1:N-1] = -np.exp(-x[1:N-1])+np.exp(-2*x[1:N-
    1]) - 2*k*x[1:N-1] + k*(x[0:N-2] + x[2:N])
    dudt[N-1]= -np.exp(-x[N-1])+np.exp(-2*x[N-
    1]) - k*(x[N-1]-x[N-2])

    return np.append(dxdt,dudt)

def ac_DW(t,xu,k):

    x, u = xu[:N], xu[N:] # Split xu vector into x and
    u. #
    dxdt = u # New position.
    dudt = np.zeros((N,1)) # New velocities.
```

```
    dudt[0]= (x[0]-1)-(x[0]-1)**3 - k*(x[1]-x[0])
    dudt[1:N-1] = (x[1:N-1]-1)-(x[1:N-1]-
    1)**3 - 2*k*x[1:N-1] + k*(x[0:N-2] + x[2:N])
    dudt[N-1]= (x[N-1]-1)-(x[N-1]-1)**3 - k*(x[N-1]-
    x[N-2])

    return np.append(dxdt,dudt)
```

Linearising equation of motion for finding the tangent map for hard φ4, morse and double well non-linear potentials respectively

```
def bre_hard(t, exu,k) :

    x, u = exu[:N].reshape(N,1), exu[N:2*N].reshape(N,1
    ) # Split xu vector into x and u.#
    dxdt = u # New position.
    dudt = np.zeros((N,1)) # New velocities.
    dudt[1:N-1] = -x[1:N-1]-x[1:N-1]**3 - 2*k*x[1:N-
    1] + k*(x[0:N-2] + x[2:N])
    dudt[0]= -x[0]-x[0]**3 - k*(x[1]-x[0])
    dudt[N-1]= -x[N-1]-x[N-1]**3 - k*(x[N-1]-x[N-2])

    ee=exu[2*N:].reshape(2*N,2*N)

    dm=[]

    for i in range(2*N):

        ex, eu = ee[i,:N], ee[i,N:] # Split xu vector i
        nto ex and eu.
        ex, eu = ex.reshape(N,1) , eu.reshape(N,1)

        dexdt = eu # New position.
        deudt = np.zeros((N,1)) # New velocities.
```

```
        deudt[1:N-1] = -(1+3*x[1:N-1]**2)*ex[1:N-
        1]- 2*k*ex[1:N-1] + k*(ex[0:N-2] + ex[2:N])
        deudt[0]= (-1-3*x[0]**2)*ex[0] - k*(ex[1]-
        ex[0])
        deudt[N-1]= (-1-3*x[N-1]**2)*ex[N-1] - k*(ex[N-
        1]-ex[N-2])
    det=np.append(dexdt,deudt)
    dm.append(det)

    dm=np.array(dm).flatten()

    dxut=np.append(dxdt, dudt)
    det=np.append(dexdt,deudt)

    return np.concatenate((dxut, dm), axis=0) # Concate
    nate the two vectors.


def bre_morse(t, exu,k) :

    x, u = exu[:N].reshape(N,1), exu[N:2*N].reshape(N,1
    ) # Split xu vector into x and u. # Split xu vector
     into x and u.#
    dxdt = u # New position.
    dudt = np.zeros((N,1)) # New velocities.
    dudt[0]= -np.exp(-x[0])+np.exp(-2*x[0]) - k*(x[1]-
    x[0])
    dudt[1:N-1] = -np.exp(-x[1:N-1])+np.exp(-2*x[1:N-
    1]) - 2*k*x[1:N-1] + k*(x[0:N-2] + x[2:N])
    dudt[N-1]= -np.exp(-x[N-1])+np.exp(-2*x[N-
    1]) - k*(x[N-1]-x[N-2])

    ee=exu[2*N:].reshape(2*N,2*N)

    dm=[]

    for i in range(2*N):

        ex, eu = ee[i,:N], ee[i,N:] # Split xu vector i
        nto ex and eu.
```

```
        ex, eu = ex.reshape(N,1) , eu.reshape(N,1)

        dexdt = eu # New position.
        deudt = np.zeros((N,1)) # New velocities.
        deudt[1:N-1] = -(-np.exp(-x[1:N-1])+2*np.exp(-
        x[1:N-1]))*ex[1:N-1]- 2*k*ex[1:N-
        1] + k*(ex[0:N-2] + ex[2:N])
        deudt[0]= -(-np.exp(-x[0])+2*np.exp(-
        x[0]))*ex[0] - k*(ex[1]-ex[0])
        deudt[N-1]= -(-np.exp(-x[N-1])+2*np.exp(-x[N-
        1]))*ex[N-1] - k*(ex[N-1]-ex[N-2])

    det=np.append(dexdt,deudt)
    dm.append(det)

    dm=np.array(dm).flatten()

    dxut=np.append(dxdt, dudt)
    det=np.append(dexdt,deudt)
    return np.concatenate((dxut, dm), axis=0) # Concate
    nate the two vectors.
```

```
def bre_DW(t, exu,k) :

    x, u = exu[:N].reshape(N,1), exu[N:2*N].reshape(N,1
    ) # Split xu vector into x and u. # Split xu vector
     into x and u.#
    dxdt = u # New position.
    dudt = np.zeros((N,1)) # New velocities.
    dudt[0]= (x[0]-1)-(x[0]-1)**3 - k*(x[1]-x[0])
```

```
dudt[1:N-1] = (x[1:N-1]-1)-(x[1:N-1]-
1)**3 - 2*k*x[1:N-1] + k*(x[0:N-2] + x[2:N])
dudt[N-1]= (x[N-1]-1)-(x[N-1]-1)**3 - k*(x[N-1]-
x[N-2])

ee=exu[2*N:].reshape(2*N,2*N)

dm=[]

for i in range(2*N):

    ex, eu = ee[i,:N], ee[i,N:] # Split xu vector i
    nto ex and eu.
    ex, eu = ex.reshape(N,1) , eu.reshape(N,1)

    dexdt = eu # New position.
    deudt = np.zeros((N,1)) # New velocities.
    deudt[1:N-1] = (1-3*(x[1:N-1]-1)**2)*ex[1:N-
    1]- 2*k*ex[1:N-1] + k*(ex[0:N-2] + ex[2:N])
    deudt[0]= (1-(3*x[0]-1)**2)*ex[0] - k*(ex[1]-
    ex[0])
    deudt[N-1]= (1-(3*x[N-1]-1)**2)*ex[N-
    1] - k*(ex[N-1]-ex[N-2])
    det=np.append(dexdt,deudt)
    dm.append(det)

dm=np.array(dm).flatten()

dxut=np.append(dxdt, dudt)
det=np.append(dexdt,deudt)
return np.concatenate((dxut, dm), axis=0) # Concate
nate the two vectors.
```

Main algorithm for creating breathers

```
dx_1=0.1
N=40
xu=np.empty((6,80))
points=[]
period=[]

for x in range(6):
  print(x)

  xu[x],points_in_period,Period,w=Ampl_freq(ac_hard,dx_
  1,N)
  points.append(points_in_period)
  period.append(Period)
  dk=0.001

  for k in range(120):
    dk=float("{:.3f}".format(dk))

    #print(dk)

    before=np.zeros(2*N)
    before+=xu[x]

    ee=np.identity(2*N)
    ee=ee.flatten()
    exu=np.concatenate((xu[x],ee))
    exu=exu.flatten()

    sol = solve_ivp(bre_hard, t_span = (0, Period), y0
    = exu, args=(dk,),
    t_eval = np.linspace(0,Period, points_in_period +1
    ), vectorized = True)

    m=np.zeros((2*N,2*N))
    m=sol.y[2*N:,-1].reshape(2*N,2*N)
    #m=m.reshape(2*N,2*N)
    xu[x]=sol.y[:2*N,-2]
```

```
    a=c=np.zeros(N)
    a=m[:N,:N]
    c=m[N:,:N]
    a=a-np.identity(N)
    Invac=np.zeros((N,N))
    Invac+=la.inv((np.dot(a,a.T)+np.dot(c,c.T)))
    ac_tran=np.zeros(N)
    ac_tran=np.append(a.T,c.T,axis=1)
    dx=np.zeros(2*N)
    dx[:N]+= xu[x][:N] - before[:N]
    delta=np.zeros(N)
    delta-=Invac@ac_tran@dx
    #print(delta)
    xu[x][:N]=(before[:N]+delta)
    xu[x][N:]=0

    dk+=0.001

  dx_1+=0.01
```

Preparing the data and train a CNN

```
#Spliting our Data to train and test set with an 80%-
20%

X_train, X_test, labels_train, labels_test = train_test
_split(

BP_inputs.reshape((449,40,822,1)), BP_labels, test_size
=0.20, random_state=42)

#Catecorizing our Labels

train_labels = to_categorical(labels_train)

test_labels = to_categorical(labels_test)
```

```
#Normalizing the Image Data we created for train our CN
N

X_train=datagen.flow(X_train, y=None,batch_size=len(X_t
rain),shuffle=None).next()
X_test=datagen.flow(X_test,y=None,batch_size=len(X_test
),shuffle=None).next()


#Creating our CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation = 'relu', input
_shape = (40, 822,1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(Flatten())
model.add(Dense(64, activation = 'relu'))
model.add(Dense(2, activation = 'softmax'))

model.summary()

model.compile(optimizer = 'adam',
              loss = 'BinaryCrossentropy',
              metrics = ['accuracy'])

es = EarlyStopping(monitor='val_loss', mode='min', verb
ose=1, patience=5)
mc = ModelCheckpoint('best_model.h5', monitor='val_loss
', mode='min', save_best_only=True)

history = model.fit(X_train,train_labels,validation_spl
it=0.2,
                    epochs = 100, callbacks=[es,mc])
```

Python Libraries

- Numpy
- Scipy
- Sklearn
- Matplotlib
- Tesnorflow version 2.8.2
- Keras

Useful packages were

- solve_ivp from scipy.integrade
- ImageDataGenerator form keras.preprocessing