Develop a Image-to-Image Translation Model to Capture Local Interactions in Mechanical Networks (GAN)

Final Report

This report summarizes the findings of my 2nd capstone project for Springboard. It includes a recall of the motivations, the type of data handled, the methodology used and the results. While the business prospect for this project isn't evident at first, the reader must realize this work is intensively related to my physics research at the Georgia Institute of Technology, on mechanical deformations in metamaterials.

Introduction

Mechanical metamaterials are systems formed by connected building blocks, which grant them unconventional mechanical properties. How these blocks connect with one another determines the structure of the system and controls its mechanical deformations. Depending on its structure, a metamaterial can be entirely rigid or can possess soft regions. Under certain circumstances, it can convey forces, stress and displacements in a particular direction. Because the structure governs the mechanics of these materials, micro-sized metamaterials can carry functionalities identical to their human-sized counterparts, so long as they carry the same *structure*. Regardless of their size, the model for the structure of these metamaterials is the same: the blocks (also known as sites, nodes, or balls) are spread out in a roughly periodic structure and are connected by bonds (also known as edges or springs). These structures are commonly referred to as **mechanical networks**. The nature of the local interactions, i.e. the arrangement of the blocks and their connecting bonds, is commonly referred to as the *bond connectivity*, and it is critical to know in order to understand the mechanical properties of the system.

Motivation / Goal

As we've mentioned above, some of these mechanical networks are extremely small in size - of the order of the micron (μm). Consider the following situation: researchers are studying a micro-sized mechanical network and are searching for the local interactions (the bond connectivity) in order to interpret the mechanical properties of the system at stake. In principle, the researcher can identify where the sites are (the building blocks), but may struggle identifying the bonds - often enough, they are much smaller than the sites and harder to distinguish. This

project aims to design a neural network model that will automatically plot the bond connections between the sites of a given system. In particular, the goal is to build a **Image-to-Image Translation** model which inputs the image of the particles and outputs the same image with the bonds drawn:

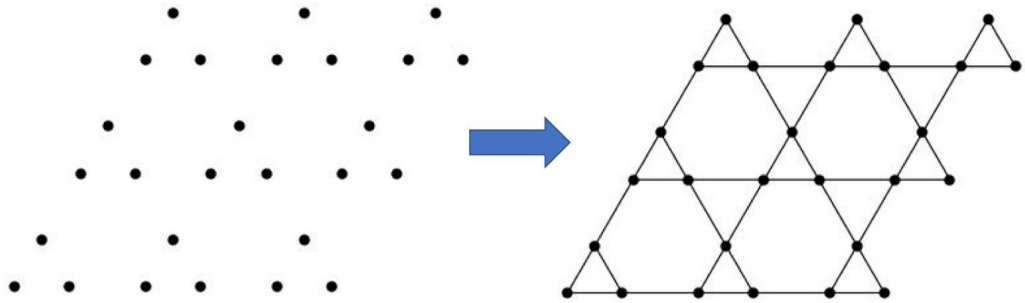


Figure 1: Ordered Kagome lattice without (left) and with (right) bond connectivity

A system often studied in the research on mechanical metamaterials is the Kagome lattice, drawn above without (left) and with (right, solid lines) the bond connectivity. On both images, the sites (black dots) are ordered periodically in space in groups of three. Each set of three neighboring sites constitute what we call a *cell*. However, in general, the systems encountered will be a bit more disordered, like in the following image.

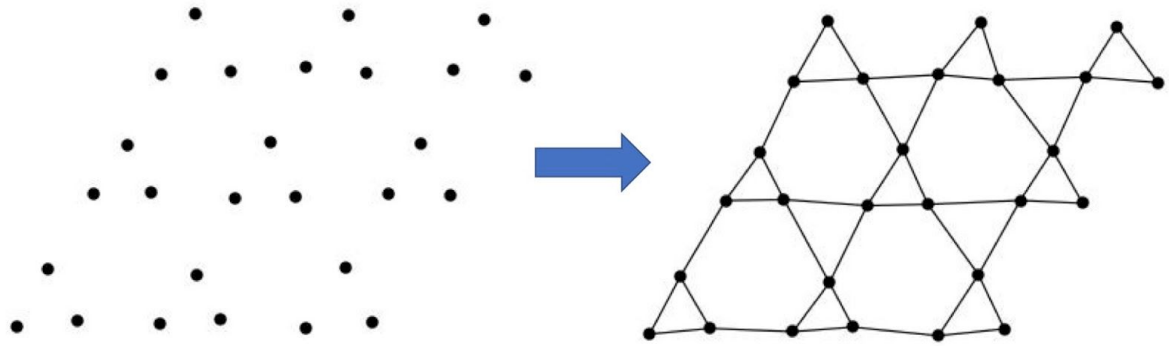


Figure 2: Disordered Kagome lattice without (left) and with (right) bond connectivity

As observed, the input image of figure 2 appears identical to the one of figure 1, at least to the human eye. But despite their apparent similarities, when the bonds are plotted, it becomes clear each image represents two very different systems. This “bond-plotting” model can facilitate the classification of these mechanical systems and could be very enriching for the scientific community that studies such materials.

The Data

The data used in this project is generated through my research work, using the program *Mathematica*. While these images are artificial, they correspond to the structure of metamaterials that are frequently studied within the research community for their exciting mechanical properties. The images are generated in jpg format and are (256 x 256) pixels in size.
On these images, we can distinguish about 30 sites (the black dots) and about as many bonds.

Generating the data via Mathematica

For each figure, the points on a Kagome structure are generated as follow:

- The first cell (the original 3 points) is drawn in the lower-left corner.
- The remaining cells are generated by a set of discrete geometric translations to form a periodic structure. This is the ordered structure of figure 1.
- Each point in each cell is then shifted by a small and yet random displacement to create an image like figure 2: the structure appears to carry a long range order but contains small variations from one cell to another.
- For the images with bonds, the points are then connected by straight lines (the edges) as indicated on figure 1 and 2. The bond connectivity is the same for all figures.

The practical advantage here is the potential to easily generate 1000s of pictures, without the need to further edit them. Therefore, the wrangling and cleaning steps of this project are small, if not non-existent.

More Data

Per my previous milestone reports, we've seen that the model works well in generating images for Kagome structures (images like those of figure 1 and 2). Therefore, our final approach consists in building a model capable of drawing the bond connectivity for totally different sets of images, such as the triangular lattice (same as the Kagome lattice, except only triangles) or the square lattice (four sites per cell). To that end, we generate more training data, with the hope that the model will be able to distinguish between the different systems considered (Kagome vs square vs triangular) and still plot the appropriate bonds.

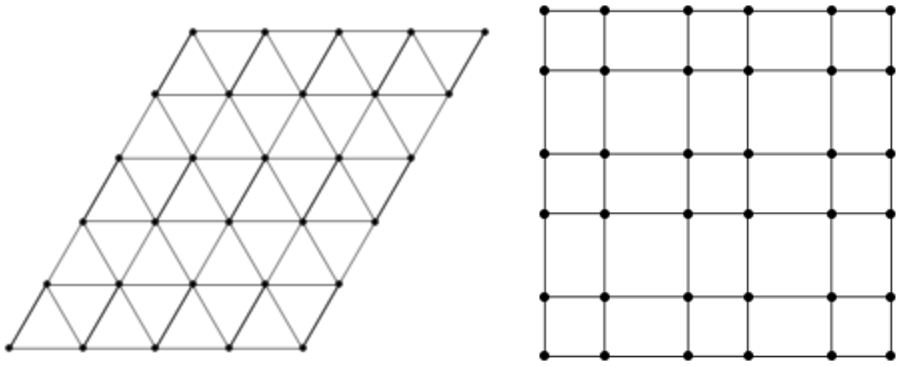


Figure 3: Ordered triangular lattice and square lattice

Loading data and verification

All jpg images are then uploaded on my Google Drive, which I mount to the Google Collab notebook in order to build, train and test the deep learning model. The images are loaded onto the notebook using the Keras.preprocessing.image function and the pixels are stored in a 3D numpy array, the first two dimensions corresponding to the image size, the last one being the RGB code for each pixel (ranging from 0 to 255). A good way to make sure the images are properly deconstructed is to reconstruct the pixel arrays and to display the corresponding images using imshow from numpy. For the sake of terminology, a source image is an image with dots only; a target image is the image with the bonds; a generated image is the output of the generator.

Methodology

The deep learning model used to generate an image-to-image in this project is a **Generative Adversarial Network** (GAN). The Pix2Pix model developed by Dr. Jason Brownless for Machine Learning Mastery (and referenced here) serves as a basis for my code and I thank Jason for sharing his work with the rest of the machine learning world.
https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

As mentioned earlier, except for generating the data, all operations are done and all code is written on Google's Colab notebook, a practical tool which allows its users to take advantage of Google's computer resources, such as their cloud-hosted GPUs. This type of computer power is needed to run the GAN, because of its ability to process and train the billions of parameters that make for the structure of the neural network.

Summary of GAN

The GAN's architecture consists of two main models that train together in an adversarial manner - the discriminator and the generator - along with other features that bind and train them together. Through this section, I describe the architecture of the GAN and some of the parameters I choose in order to build and train the model

The **discriminator**:

The discriminator is a deep Convolutional Neural Network (CNN) that takes an image shape and performs **conditional-image** classification. In summary, it takes the source image (without bonds) and the target image (with bonds) and classifies if the target is the *real* translation of the source. In detail, it :

- creates source/target inputs with same shape as image_shape,
- merges source/target by concatenation so that the input of the deep convolutional neural network has dimensions 256*256 and 6 channels (3 for each source/target),
- takes the merged input and makes it go through the CNN,
- compiles the output of the CNN and uses it as model.

The deep CNN consists of 6 layers.

The first layer takes as input (x, 256, 256, 6) and makes it go through 64 filters, so that the output is (x, 128, 128, 64). x corresponds to the number of input images, while the last element is the number of channels. Through the same process, the next layers transforms the inputs from (x, 128, 128, 64) to (x, 64, 64, 128) to (x, 32, 32, 256) to (x, 16, 16, 512) and finally to (x, 16, 16, 1).
Each layer uses filters of size (4,4), strides (2,2) and the feature 'same padding' so that the pixels on the borders are equally considered. The layers are connected by Batch Normalization and ReLU function in order to keep the channels normalized. The last output is activated via sigmoid channel.

The weights of each layer are initially randomly distributed around 0. The key feature is that the weights update for the discriminator needs to be slow compared to the generator, hence the use of binary cross entropy with a weighting of loss_weights=0.5 so that the updates are twice as slow.

A **generator**:

It generates an image. The generator follows a U-net architecture and consists of three sections: the encoder (contraction), the bottleneck and the decoder (expansion).

1) The encoder section consists of multiple blocks that downsample the input image. Each block is a convolution layer with an ever-so-increasing number of filters which allows the

architecture to learn the complex structure of the image effectively. The encoder takes a (x, 256, 256, 3) image and outputs one with (x, 2, 2, 512) through several layers.

2) On the contrary, the decoder consists of several expansion blocks, with a number of filters ever-so-decreasing. Importantly enough, the input of each expanding layer features both the output of the previous expanding block AND the output of corresponding contracting block. This ensures the features learned during the encoding phase are re-used to reconstruct the image in the decoding one. Therefore, there are as many layers in the encoder than in the decoder section: 7.
3) The bottleneck section connects the encoding and decoding sections.

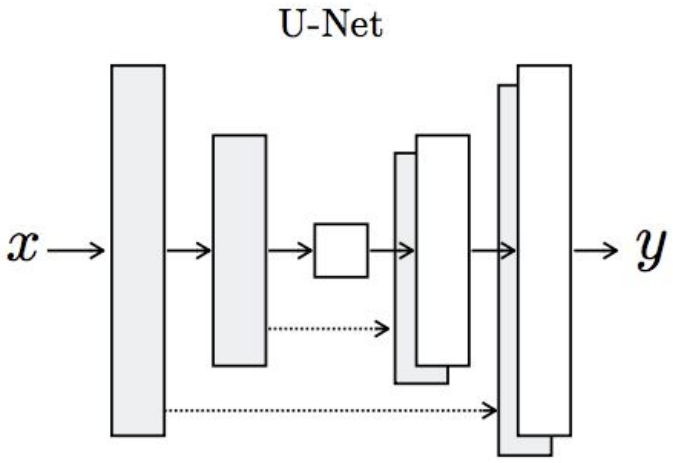


Figure 4: U-Net Architecture of the Generator Model

The final layer outputs an image with size (x, 256, 256, 3) identical in shape to those that were fed to the generator.

The **Composite:**

The Generative Adversarial Network (GAN) is a composite model which consists of both discriminator and generator at once. While the generator updates its weights via the discriminator, the discriminator isn't trained and updates its weights "on its own". Note, the updates for the composite model are done with respect to two targets:

- the output of the discriminator (to find if the generated images are real). This is done via cross entropy loss ('binary_crossentropy') and forces large weight updates to the generator
- the executed real translation of the image compared to the generated one (L1 Loss)

The **train model:**

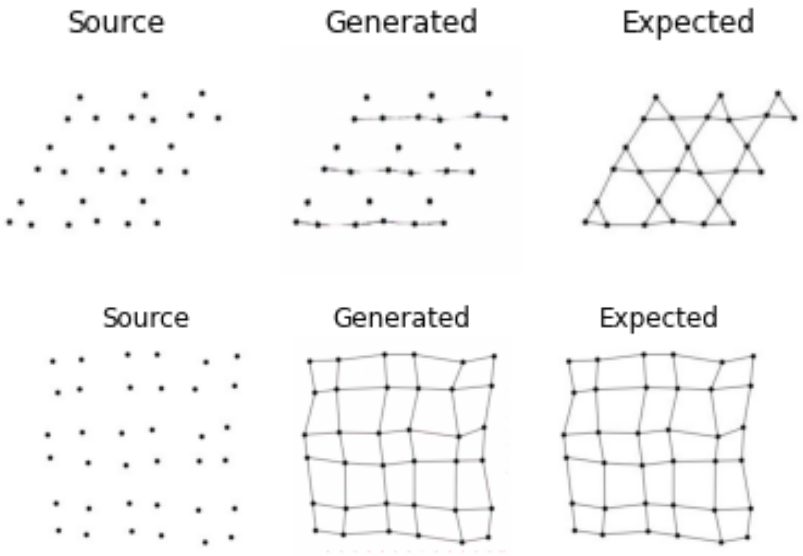
Each training step consists of using the composite model along with “real” images from the training data and “fake” generated images from the generator. The idea is that a combination of real target and fake generated images, respectively classified as 1 and 0 will help the generator build a model that converges faster toward a solution.

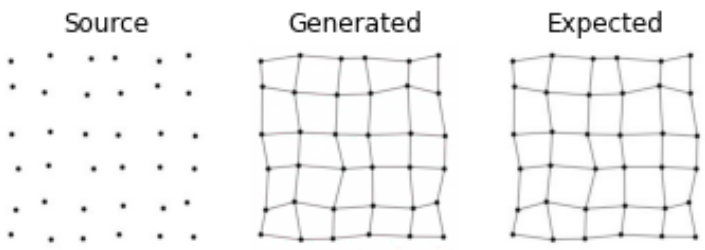
Results

Our final model uses as training data:

- a dataset of size 90, 30 images from each category (Kagome, square, triangular)
- the number of patch (which is the size of each image and which should be equal to the output shape of the discriminator)
- the number of batches (3), meaning there are 30 images drawn at random in each batch per epoch
- 500 epochs; this means there are a total of 500*30 = 15000 iterative steps. The run time, despite using Google’s GPU’s, is about 2 hours along!

We summarize the performances of the model every 300 iterations. The model itself is saved in a h5 file, while a generated image drawn at random from the training data is plotted against its source and its target images. We display the results (generated vs source and expected) for 3 models and notice the evolution of the drawing of the bonds





From top to bottom: evolution of a generated image from the training set through training iterations (6300 / 15000 -> 11200 / 15000 -> 15000 / 15000)

Since this is the training data, we expect the final model to work well. What's even more exciting is the model runs well against a validation set. One of the previous built models, trained on Kagome structures only, generated the following images for all 3 validating samples.

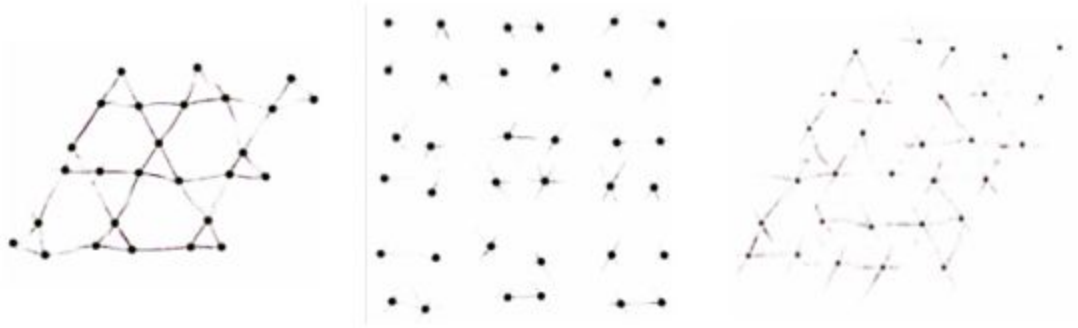


Figure 6: Validating model trained on Kagome samples with Kagome (left), square (middle) and triangular (right) lattices

The **new, more inclusive** model however generates the following

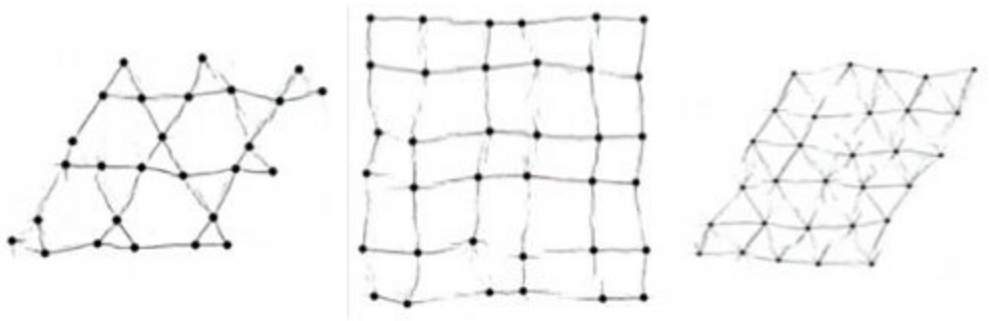


Figure 7: Validating model trained on all 3 samples with Kagome (left), square (middle) and triangular (right) lattices

Conclusion and Follow up

By training on 90 pairs of images only, the model can take unseen mechanical networks and plot the proper bond connectivity. Importantly enough, the model is well built to detect the position of the sites themselves (the black dots on the images), which remain unchanged on the generated outputs. This is critical because the positions of the blocks play an important role in the mechanical properties of the metamaterial: even if the plotting of the bond connectivity is correct, if the position of the sites is changed, we may find ourselves with a totally different metamaterial. Evidently, it is important that the fundamental properties of the system remain unaltered when processed by the algorithm.

The other significant result is the ability of the model to not only properly plot the bonds, but to identify which of the three systems is considered: the Kagome, the square of the triangular lattice. This is significant because in the context of this research on mechanical metamaterials, the challenge researchers may have in categorizing unseen systems may be tackled with this so-called “classification” model, classification in that it visually indicates which structure we’re dealing with.

While this is a satisfying result, there are things about the model that can be improved

- If the training is run longer (more epochs) and on a larger dataset, we would expect the plotting of the bonds to be thicker and less sparse at some locations
- One thing I’ve alluded to in the previous paragraph is the ability of the model to *classify* which of the categories the system considered belongs to. It is certainly feasible to turn this into a “true classification” problem, where one can expect to identify the system rigorously, via a metric, and not just visually.
- Metamaterials are not limited to the three systems we’ve covered. It would be interesting to see how more complex structures can be handled by the model.