Machine learning for solid mechanics: stress prediction using a 3D U-Net

Semester project by Paul Devianne
***Supervisors**: Lucas Fourel, Lori Graham-Brady*
Computational Solid Mechanics Laboratory – LSMS
Swiss Federal Institute of Technology – EPFL

Abstract
Simulations of materials physics at the micro-scale are useful for numerous applications. Recently, machine learning methods have been developed to replace numerical schemes due to their fast compute time and versatility. This project presents a deep learning model for predicting stress distributions in 3D solid materials, inspired by a prior 2D study [1]. Our approach employs a 3D U-Net deep convolutional neural network (CNN) trained to map the spatial arrangement of microstructural features to the corresponding 3D stress tensor fields. The 3D U-Net model provides local stress tensor fields for down-scaling. We demonstrate that the method achieves similar error rates to the 2D approach. We developed data augmentation techniques which improved performance of the model. We found that the model scales well to finer resolutions. We also performed hyper-parameter tuning.

1 Introduction

With the explosive growth of available data and computing resources, recent advances in deep learning and data analytics have yielded transformative results across diverse scientific disciplines, including image recognition, genomics, or text generation. In parallel, differential equations can be numerically integrated to solve physical models. These numerical methods represent an immense source of information for training a machine-learning (ML) model. Hence, the computational cost of numerical methods for physical problems can be replaced by a pre-trained ML-model.

The microscale behavior of solid materials controls the macroscale response. Critical damage such as crack initiation in composite materials are associated with high microscale stresses in the material. The goal of multiscale methods is to connect the microscopic features to the macroscopic material properties [2], [3]. At the microscale, computational time can become expensive depending on the complexity of the microstructure.

Recently, ML-based methods have been developed as surrogate model in solid material simulations [4], [5]. Lately, there has been development in using ML methods for predicting full-field microscale stresses in heterogeneous material microstructures [6], [7]. Other works have considered 2D heterogeneous microstructures as images using convolutional techniques for predicting stress maps. These models have demonstrated a significant accuracy for elastic materials under a general load state or complex microstructure geometry [1], [8]. The research project presented here aims at extending these results to the 3D problem.

For this project, we developed a deep-learning based method which learns from FFT-generated 3D microstructures and the associated stress tensor under a specific loading. We study a linear elastic material with spherical inclusions to introduce heterogeneity. The goal of the project is to study the computational cost and the accuracy of the 3D ML-model. Inherent physical symmetries of the problems make data augmentation possible. Additionally, this report presents a hyper-parameter tuning of the model. Parallelization of both training and inference of the model are studied. Deep learning models are promising solutions for accelerating computationally expensive simulations.

2 Methods

This section presents the different experiments conducted during the semester project.

2.1 Physical problem & symmetries

For this research, we study an isotropic elastic heterogeneous material. We introduce a stress tensor field $\sigma(\vec{x})$ at the location $\vec{x} \in \Omega$, where Ω is the space of the microstructure. The microstructure is subject to a strain tensor $\varepsilon(\vec{x})$. Elastic material in 3D verifies Hook's law which in Cartesian coordinates writes as:

$$\sigma_{ij} = c_{ijkl}\varepsilon_{kl} \tag{1}$$

with c_{ijkl} the stiffness tensor. Inherent symmetries of the problems, reduces the number of stress and strain components to 6: $\sigma_{ij} = \sigma_{ji}$. The ML model aims at predicting the stress physically defined by an equation $\sigma_{ij} = f(\varepsilon_{kl})$. In the study, we will focus on predicting a single component of the microscopic stress tensor σ_{xx} under a single uniaxial loading with macroscopic strain $\bar{\varepsilon}_{xx} = 1$. Using the same approach each of the 6 stress components could be determined. Additionally, the linearity allows superimposing stress fields from different loading conditions.

Furthermore, we will assume that, if we are able to build an efficient model for one of these eight equations, we should be able to build a similar model for the other equation. A generalized model would use the superposition principle to summarize the result for a general loading. For the rest of this study we will focus on a uni-axial stress σ_{xx} under a compression on the same axis $\bar{\varepsilon}_{xx}$.

2.2 Dataset

The dataset for the 3D ML model comprises 768 independent samples, each represented as 3D dimensionless cube with a given resolution. The input elements feature normalized Young's modulus values. The matrix has a modulus of 1 and 100 spherical inclusions with modulus of 0.1 positioned randomly within the matrix. Diameters vary from 0.1 to 0.2 following a uniform distributions. All samples have a Poisson ratio of 0.3 and are subjected to periodic boundary conditions. The output for each input microstructure image is the corresponding σ_{xx} stress tensor under compression $\bar{\varepsilon}_{xx}$ for each pixel, calculated using an FFT numerical method. An example of input/output is displayed on the upper part of Figure 1.

The dataset is available with two resolutions: 64 or 128 voxels per dimension.

2.3 Data Augmentation

We study the effects of data augmentation on the model. Data augmentation helps address the challenge of a limited dataset. When training data is acquired experimentally or using expensive numerical simulations, data augmentation can present a significant benefit. Identifying the symmetries of the problem is essential. The boundary conditions are periodic and the material isotropic, allowing us to apply transformations such as rotations of 180° around any of the three axes (i.e., $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$ where $\theta = 180$), which produce the same stress σ_{xx} under compression $\bar{\varepsilon}_{xx}$. Flipping the structure along a middle plane is another viable transformation. We will investigate the impact of these various transformations on the training process. See Figure1 for more details.

2.4 Deep Learning Model

Artificial neural networks are useful tools to model a continuous function from a limited number of samples (see Figure 3). A non-linear combination of the input values are accumulated at neurons in the next layer. The combination is based on an activation function, and weight parameters. After multiple layers, the network make a prediction y_{pred} on the output of the model given the input x. The distance to the true value y_{true} is evaluated through a loss function $\mathscr{L}$. To minimize the error, we make use of the back-propagation of the loss gradients: an optimizer changes the combination of the neuron values starting from output to input layer, to minimize the loss/error function.

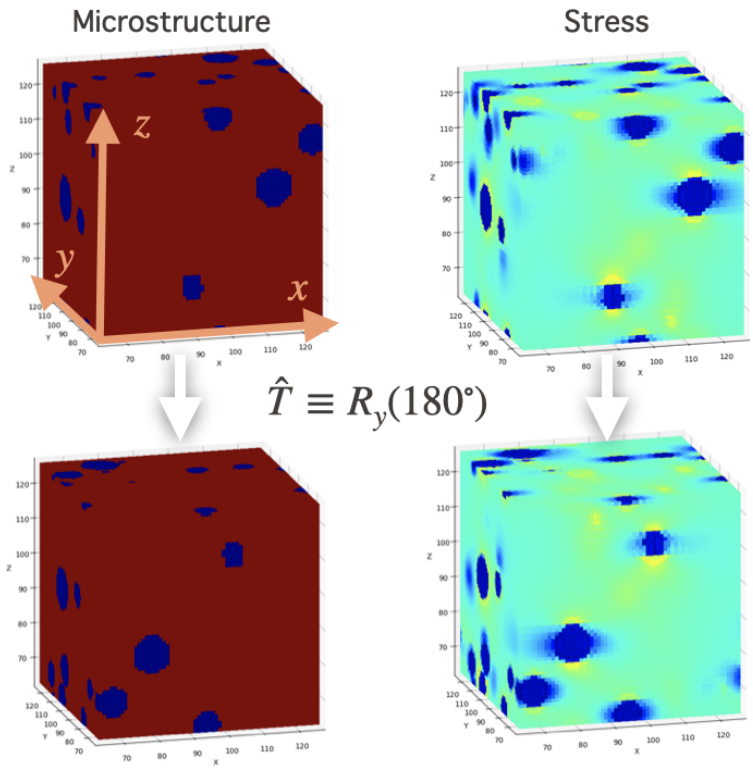


Figure 1: Example of data augmentation by a rotation of 180° along the y-axis

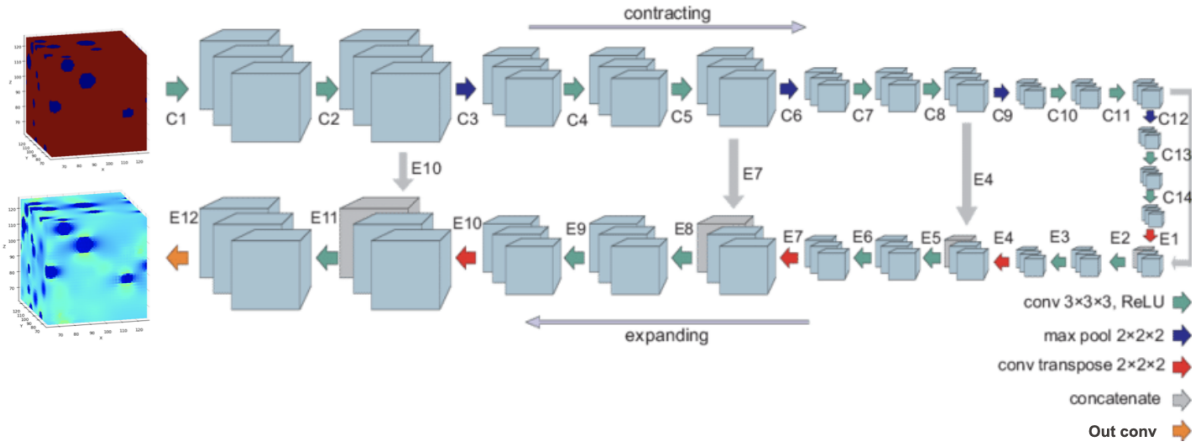


Figure 2: U-net architecture. Number of channels is not represented precisely. Adapted from Salehi et al. (2017) [10]

For datatype such as images or microstructures, the spatial location of each input value (pixels) should be accounted for. A convolutional layer aggregate neighbouring pixels to generate a filter of the input, a channel. A pooling operation down-samples the transformed input to capture features at different scales of the image.

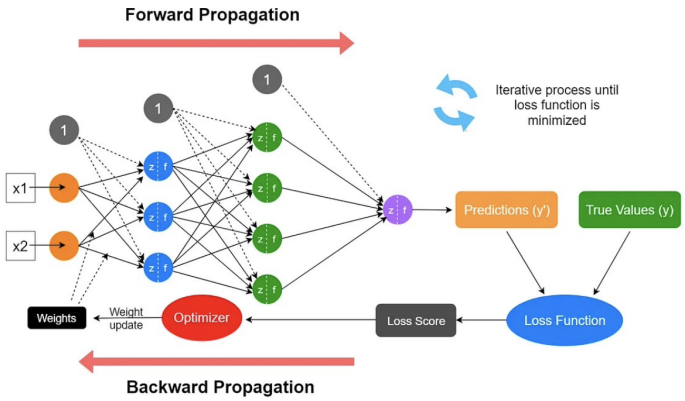


Figure 3: Training and testing curves for the optimal model.

The machine learning model used in our study is a U-net, an encoder-decoder architecture developed for computer vision tasks [9]. The U-net consists of successive convolutions and pooling operations, leading to contractions of the transformed input and subsequent expansions to the output. A key feature of the U-net is the skip-connections between layers of similar size, which help recover fine-grained details in the prediction (see Figure 2).

The network parameters are defined as follows: the number of channels starts at 4 and is then multiplied by 2 before each pooling. There is an extra convolution step between each convolution and the pooling, resulting in the channel sequence: 1-4-8-16-32-64, and then it decreases back to 1 during the expanding steps. The kernel size for convolutions is 3, stride of 1, same padding, and circular padding mode. The max-pooling is set with a kernel size of 2 and a stride of 2.

2.5 Training

The training process employs a mean squared error (MSE) loss function, which penalizes extreme values. The MSE is calculated as

$$\mathscr{L} = \frac{1}{N}\sum_{i}(\sigma_i^{(\text{pred})} - \sigma_i^{(\text{true})})^2 \tag{2}$$

with i the index of the pixel of the block, $\sigma_i^{(\text{pred})}$ the predicted stress value and $\sigma_i^{(\text{true})}$ the target value. Batch updates are performed with a default size of 64, using the Adam optimizer with a learning rate of 1×10^{-3}. The dataset is split into training and testing sets with a ratio of 75:25. We set a special maximum number of epochs depending on the experiment we want to conduct. We can also fix a minimum criterion to stop the training if the training loss is below a fixed threshold.

2.6 Hyperparameter tuning

To study the effects of the different hyperparameters on the training we will test different values and compare them. For fairness regarding network weight initialization and dataset, we will change the seed multiple time for each hyperparameter values. This randomizes the batches composition in the training set, and the network weights using Pytorch default network initialization procedure. We will train the model using these hyperparameters and compare the performance for the different values tested. To conduct the experiments we record different runs using an experiment tracking library in Python: MLflow [11]. The interactive user interface makes it practical for run comparisons. The implementation of the tracking in the code was part of the work for this project. An example of the interface is displayed on Figure 4.

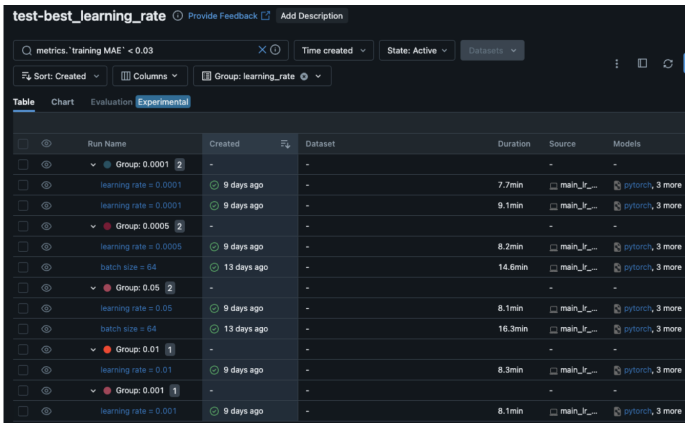


Figure 4: Example window of mLflow user interface

3 Results & Discussions

The different experiments conducted are run using the default hyperparameters on Table chosen from an *a priori* study. These values will be used unless specified otherwise.

Hyperparameter	**Value**
Stop criterion Test MAE	0.02
Optimizer	Adam
Batch size	16
# Samples (training+test set)	384
Resolution	64^3

Table 1: Default hyperparameters of the model

3.1 Hardware & parallelisation

Various devices were used for training the model, leveraging parallelisation to accelerate the process. Initially, training was conducted on a personal laptop. Subsequently, AWS cloud services with 32 cores were employed, followed by Google Colab, which provides limited GPU usage. Finally, the Izar EPFL cluster with GPU capabilities was utilized.

The time per epoch on different devices, for default parameters of training, is detailed in Table 2.

Device	**Time per Epoch (s)**
1 core (personal laptop)	957.12
32 cores (AWS cloud services)	295.15
GPU (Tesla V100-PCIE-32GB, Izar EPFL cluster)	2.74

Table 2: Training time per epoch on different devices

3.2 Optimal model

This section presents the best-performing model that has been trained with the most suited hyperparameter values. An overview of these parameters will be given later under a different experiment. We aim to present the lowest error the model can reach first. An example prediction of the model is displayed in Figure 5. The model was able to predict the output of the testing set with a relative Mean Relative Error (MAE) of **1.2 %** after 500 epochs of training. The model is trained using 32 cores. The total training time is 5.5h. An example of the training and testing curve can be seen on Figure 6.

This result proves the capability of the model at accurately predicting the stress, extending the results previously obtained in 2D [1]. One would need a better understanding of the error made currently by the model in order to improve it further. Figure 7 demonstrates that the error mostly comes from extreme values of stress. More precisely, the peaks observed represent the most common values to predict, or the homogeneous material areas in the microstructures where the stress gradient is null. Applications such as crack initiation require knowledge of high-value stress areas. For future development could focus on correcting these high-stress values.

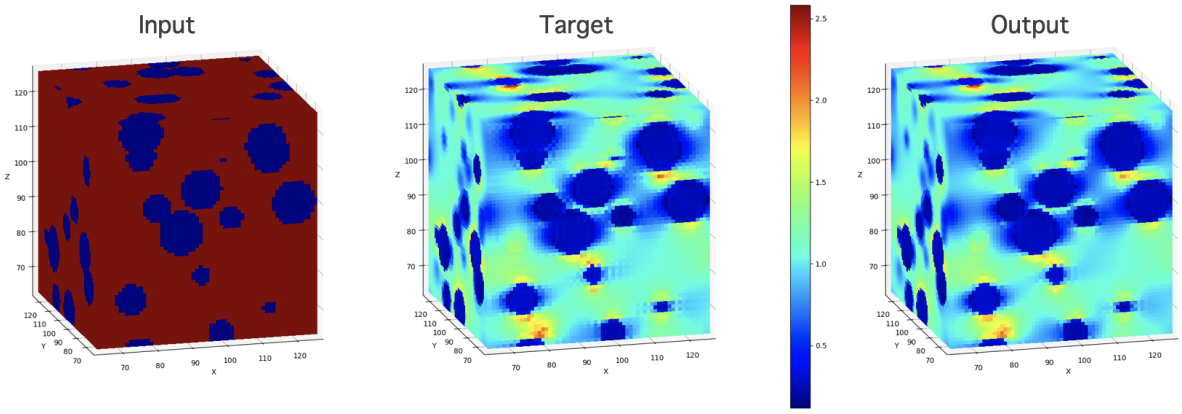


Figure 5: Worst prediction (Highest Mean Absolute Error (MAE)) of the optimal model on the testing set.

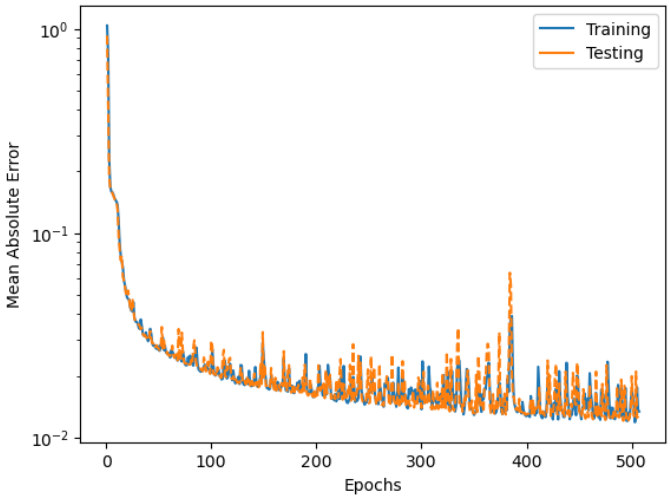


Figure 6: Training and testing curves for the optimal model.

3.3 Data Augmentation

Robustness to Transformation The question of robustness using data augmentation is not obvious in our case. For object labeling tasks, the single output makes transformation of input invariance easy to learn for the model. The higher dimensional data to predict here is more complex and data augmentation might not bring "robustness to transformations" for the model. Figure 8 presents a study on the fraction of augmented samples in the dataset. Starting with 128 independent samples, a fraction of those samples are replaced by augmented samples. The augmented samples are produced from the remaining samples by rotations or flips as explained in the previous section.

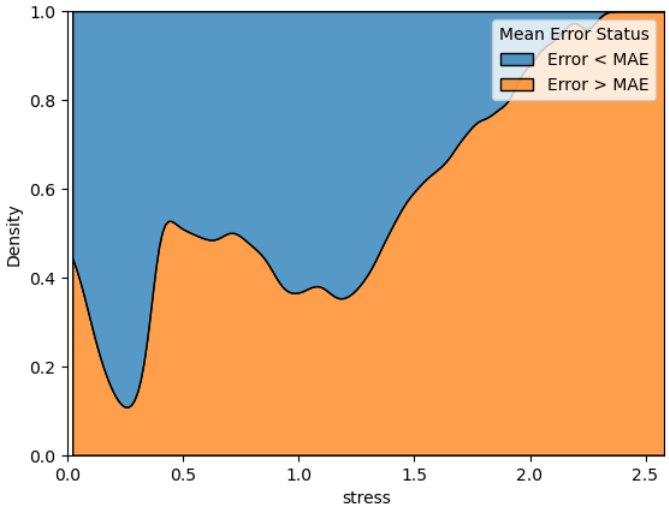


Figure 7: Worst prediction (Highest Mean Absolute Error (MAE)) of the optimal model on the testing set.

The augmentation fraction ranges from 0 to 0.5. At 0, the training set contains only independent samples. At 0.5, half of the dataset has been generated by the other half which are independent samples. The extreme value of 0.5 is to avoid having too much dependency between samples. The training of the model seems to be too sensible to the changing seed between experiments. Indeed, the results here do not show any significant advantage in using augmented samples or keeping independent samples. However, using more samples in general is an ad-

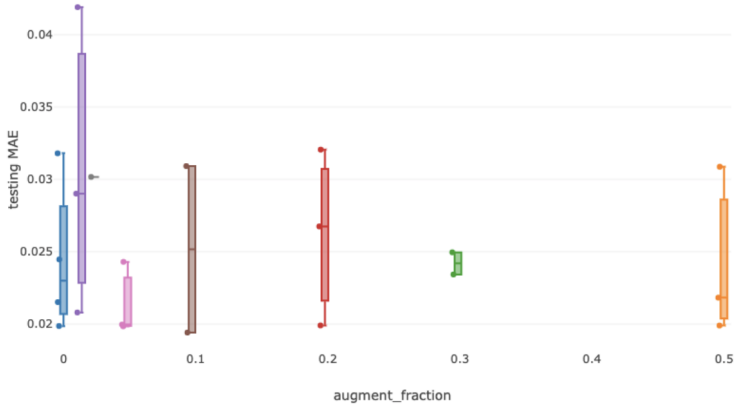
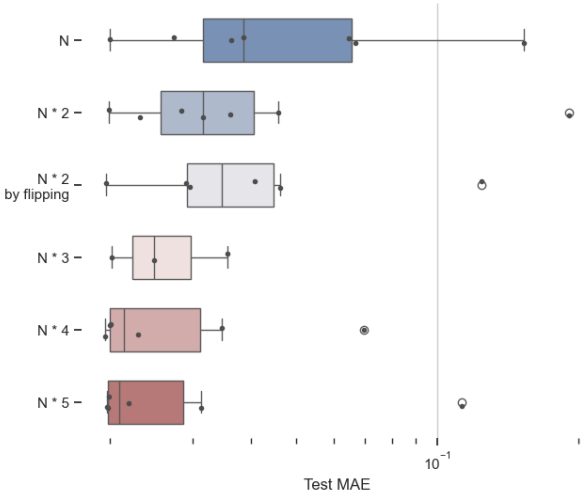
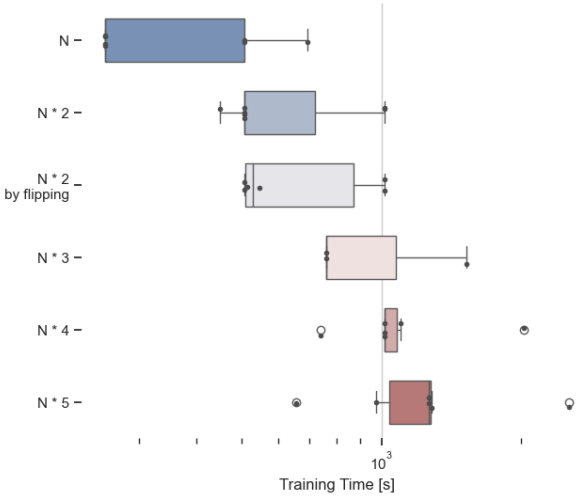


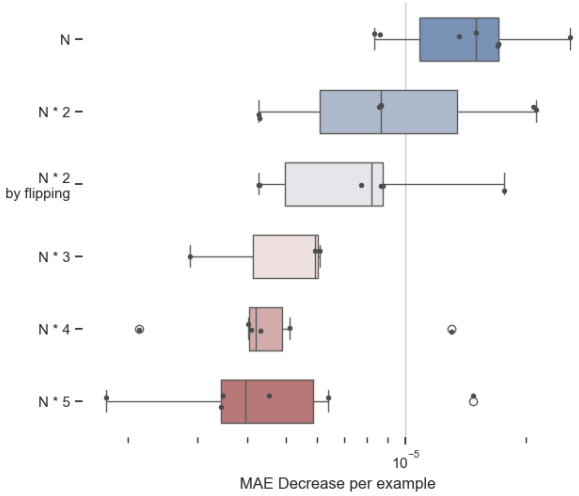
Figure 8: Performance of model (MAE) according to the augmentation fraction



(a) Testing MAE



(b) Mean Time per epoch



(c) Mean Decrease per number of examples

Figure 9: Performance of model according to the type of augmentation. The y-axis corresponds to the increase in dataset size.

vantage. Adding augmented samples on top of the training set can increase the diversity in the training set without the need of producing new samples.

Increase Dataset Size To study the effect of using data augmentation, we successively increase the size of the dataset with augmented samples and see how it impacts the error on the test set after a fixed number of epochs. Five models were trained per type of augmentation, each time on a different train/test split from a dataset of 384 samples. Failed runs (not converged) have been removed. Figure 9 a displays the result of this study. $N*2$ corresponds to rotating all samples by 180° along the x-axis and adding them to the dataset. Then we do the same for y and then z-axis. Then the augmentation is done by flipping the microstructure across the y-z plane. We obtain a dataset of size $N*5$ with $N = 128$ independent samples before augmentation.

To start with, one can observe that the more augmented sample we add to the dataset, the better the performance for the same amount of epochs. Thus, adding augmented data improves the performance. One would also notice that the models varies less between different initialization when more augmented samples are added to the dataset. Yet, looking at the points remaining outside of the boxes, one can attest that the model is very sensible to initialization. Looking at Figure 9 b, we can make two observations. First, the training time increases with the dataset size, which is logical given that there are more updates done with a bigger training set. Secondly, the more augmented samples we add, the less we gain in training time. This comes from the parallel architecture of the GPU. Thus, one would be interested in studying the efficiency of training depending on the compute time. Figure 9 c displays the error decrease depending of the number of examples learned by the model before convergence. In 1 epoch, the $N*5$ model will learn 5 times more examples than the N model. This metric is independent of the hardware used for the training. The results show that for an augmented dataset, an example will bring less information, or will decrease less the error per update. Indeed, the model has seen more examples per epochs for the augmented datasets. In general, this shows the model does not suffer from the dependency between samples. Additionally, the thin difference between the $N*4$ and $N*5$ datasets indicates that augmenting the model further is not necessary for convergence.

3.4 Hyper-parameter tuning

Similar studies have been conducted on various hyperparameters of the model such as the learning-rate, the number of samples, etc. Here we will present and analyse the studies on the batch size and the number of parameters in the network.

Batch Size The study is conducted by training the network on different batch sizes. For each size we train it 3 times on different train/test splits for robustness. The network is trained for 200 epochs, with a stop criterion when the absolute test error is below 0.02.

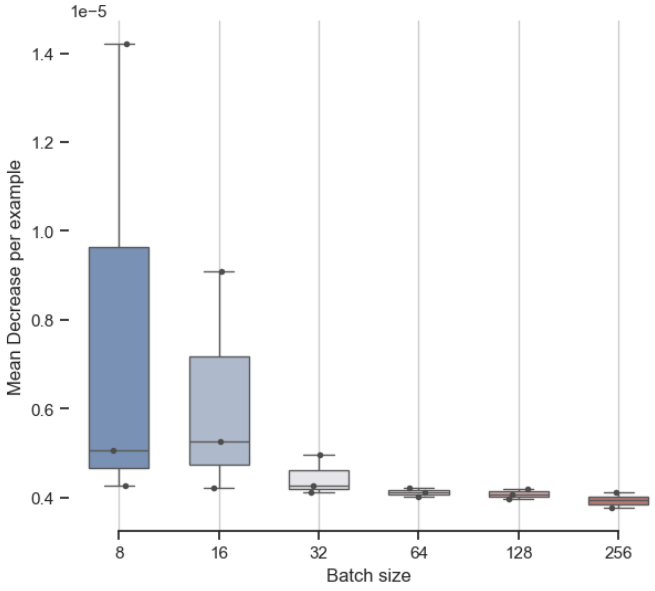


Figure 10: Performance of model (MAE) per number of examples learned by the model according to the batch size. Values of batch sizes tested $\{8, 16, 32, 64, 128, 256, 384\}$. Multiples of 2 because more suitable for the device memory.

We obtain that smaller batches reach a lower error in general. However, similarly to the case of

data augmentation, we need to compare the training times. Indeed, smaller batches usually take more time to train for the same number of epochs. There is a trade-off to find between the effort put into the training (compute time) and the performance achieved from it. Figure 10 solves the trade-off specifically for our problem.

For this study, the steepest decrease per number of examples is for the batch size of 16. This results show that the model requires sensibility to the variance between samples.

To summarise the analysis, smaller batch sizes make the model more robust to the variance between samples in the dataset. However higher batch sizes can decrease the noise during learning, weighting less the outlier samples during the updates than for higher batches. The trade-off for this study is found at 16 samples per batch.

Doubling the size of the network Increasing the size of the network usually allows for a refinement of the predictions since the model function gains in complexity. The optimal model described in the previous section is logically the model with the increased size. The increase in size comes from doubling the number of channels between each pooling step. For the network with the default number of parameters (see Methods section) the learning stops at a relative test error of 1.6%. For the model with doubled size it reaches a relative test error of 1.2%. However after reaching the plateau, the noisy learning can induce a divergence of the loss. Thus, increasing the model size still affects the performance at this scale. Depending on the application, the size of the model could be adjusted to fit the requirements in speed (smaller network) or performance (bigger network).

3.5 Increase in resolution

For application to material simulations such as multi-scale models, the model needs to perform with a comparable error on higher resolution data. Using the higher resolution dataset we train the model without changing any parameter to study the scaling. We use 128 samples from the low resolution dataset and the same samples under higher resolution so that we can compare the two training phases. Figure 11 presents the resulting prediction of the two resolution models and Figure 12 displays the training and testing MAE of each model until they reach the stop criterion.

First, one can notice the difference in scale for the two resolutions. The stress peaks at the interface between the two materials are less smoothed by the discretisation in the high res case. This already shows that for crack initiation application, where high stresses are decisive, the study of the model in higher resolutions is crucial.

Then, one can observe that the model is not overfitting since training and testing MAE curves are still upper imposed at the end of the training.

For the comparison of the two resolutions, the two models reach convergence in a comparable number of epochs: 146 for low res and 168 for high res. In general, the noise in the learning show that the convergence is reached for approximately the same number of epochs. Firstly, the size of the output for the higher resolution scales in x^3. We have $64^3 = 262144$ pixel values to predict in the low res case, and $128^3 = 2097152$ for the high res. Secondly, the number of parameters have not been changed for the high res model. This results indicates that the model scales really well to higher-resolutions, a consequently larger output without growing the network. Stress values mostly depend on neighbouring areas. This explains in the first place the use of convolutions for the task. The fact that the model scales well for higher resolutions indicates that it actually learns the local dependence between pixels, reflecting the physical equation determining the stress values.

Yet, the low error reached by the high res model has to be studied with caution. The high stress values are more localized in the high-res case. This leads to a higher volume percentage of medium stress values that the model more easily predicts. A more detailed study of the error might be necessary for future applications.

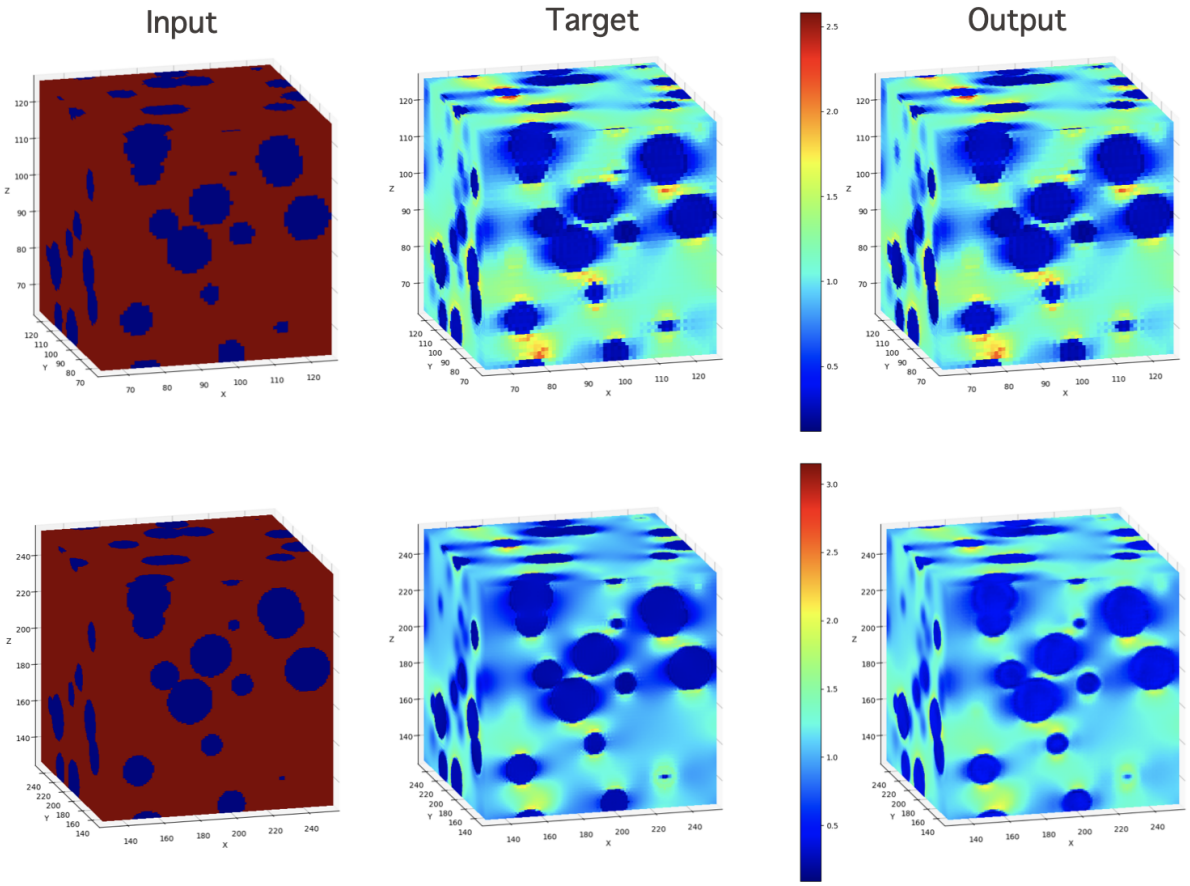


Figure 11: Example of low resolution (above) and high-resolution sample with input (left), target (middle) and model output (right) stress maps.

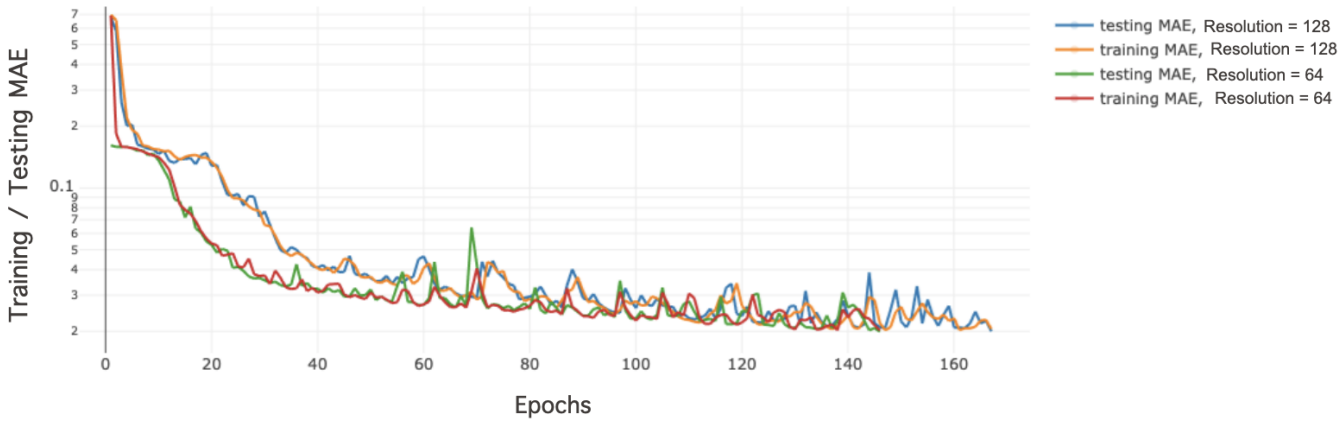


Figure 12: Comparison of MAE on training set and testing set curves for the two different resolutions 64^3 and 128^3. Stop criterion is fixed at *Test MAE=0.02*.

4 Conclusion

In this study, we have demonstrated the efficacy of a deep learning model for predicting 3D stress distributions in solid materials. By leveraging the 3D U-Net architecture and training on microstructures and their corresponding FFT-generated stress distributions, our model successfully maps the spatial arrangement of microstructural features to the corresponding stress tensor fields with high accuracy. The model achieves a mean relative error of just 1.2% with a reasonable training time of 30 minutes approximately.

Key contributions of this work include the implementation of the U-Net in three dimension. From this, we built an introduction of data augmentation techniques and the optimization of hyperparameters, leading to improved performance. Our model's ability to scale effectively to higher resolutions without a corresponding increase in computational complexity is particularly noteworthy, highlighting its potential for practical applications in material science and engineering.

Future work should focus on refining the model's handling of extreme stress values, which are critical for applications such as crack initiation prediction. Additionally, exploring more advanced data augmentation techniques and further optimizing the network architecture could yield even better performance. Finally, more complex microstructures such as polygonal inclusions and more complex behaviors such as plasticity would extend this approach to a wider range of applications. This research sets the stage for integrating advanced deep learning techniques into 3D multiscale material modeling, paving the way for faster and more accurate predictions in complex material systems.

References

[1] GUPTA, GRAHAM-BRADY, ET AL. *Accelerated multiscale mechanics modeling in a deep learning framework* (2023)

[2] J. ABOUDI, *Micromechanical analysis of composites by the method of cells.* (1989)

[3] C. HUET, *Application of variational concepts to size effects in elastic heterogeneous bodies, Journal of the Mechanics and Physics of Solids* (1990) 38 (6) 813–841.

[4] A. BHADURI, C. S. MEYER, J. W. GILLESPIE JR, B. Z. HAQUE, M. D. SHIELDS, L. GRAHAM-BRADY, *Probabilistic modeling of discrete structural response with application to composite plate penetration models*, Journal of Engineering Mechanics 147 (11) (2021) 04021087.

[5] C. RAO, Y. LIU, *Three-dimensional convolutional neural network (3d- cnn) for heterogeneous material homogenization*, Computational Materials Science 184 (2020) 109850.

[6] Z. YANG, C.-H. YU, K. GUO, M. J. BUEHLER, *End-to-end deep learning method to predict complete strain and stress tensors for complex hierarchical composite microstructures*, Journal of the Mechanics and Physics of Solids 154 (2021) 104506.

[7] R. SEPASDAR, A. KARPATNE, M. SHAKIBA, *A data-driven approach to full- field nonlinear stress distribution and failure pattern prediction in composites using deep learning*, Computer Methods in Applied Mechanics and Engineering 397 (2022) 115126.

[8] SHOKROLLAHI ET AL. *Deep Learning Techniques for Predicting Stress Fields in Composite Materials: A Superior Alternative to Finite Element Analysis* (2023)

[9] O. RONNEBERGER, P. FISCHER, T. BROX, *U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[10] SALEHI ET AL. *Tversky Loss Function for Image Segmentation Using 3D Fully Convolutional Deep Networks* (2017)

[11] `https://mlflow.org/docs/latest/index.html`