SANDIA REPORT
SAND2020-10580
Printed March 2020

Sandia National Laboratories

Reification of latent microstructures: On supervised, unsupervised, and semi-supervised deep learning applications for microstructures in materials informatics

Anh Tran, Theron Rodgers, Tim Wildey

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

Machine learning (ML), including deep learning (DL), has become increasingly popular in the last few years due to its continually outstanding performance. In this context, we apply machine learning techniques to "learn" the microstructure using both supervised and unsupervised DL techniques. In particular, we focus (1) on the localization problem bridging (micro)structure – (localized) property using supervised DL and (2) on the microstructure reconstruction problem in latent space using unsupervised DL.

The goal of supervised and semi-supervised DL is to replace crystal plasticity finite element model (CPFEM) that maps from (micro)structure – (localized) property, and implicitly the (micro)structure – (homogenized) property relationships, while the goal of unsupervised DL is (1) to represent high-dimensional microstructure images in a non-linear low-dimensional manifold, and (2) to discover a way to interpolate microstructures via latent space associating with latent microstructure variables. At the heart of this report is the applications of several common DL architectures, including convolutional neural networks (CNN), autoencoder (AE), and generative adversarial network (GAN), to multiple microstructure datasets, and the quest of neural architecture search for optimal DL architectures.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Machine learning (ML), including shallow learning and deep learning (DL), has shown a great potential to overcome human experts from various subject matters, including playing chess [2], shogi, go [3, 4]. As the fundamental block of DL, convolutional neural networks (CNN) are engineered with novel algorithms to smartly and automatically extract features from complicated image and object representations and deeply learn how to perform classification or regression. Superior performance is a signature of DL in most applications. In this report, we aim to bring the capability of DL towards materials science applications in two main tracks.

In the first track, which is supervised learning, we aim to develop the prototype of 3D-CNN called CPNet. The overarching objective for CPNet is to learn and predict the materials response, e.g. strain, obtained by crystal plasticity finite element model (CPFEM), of different materials and subjected to various loading conditions. We realize that the this problem is highly complicated. Therefore, within the scope of this report, we will limit to a fixed materials that is subjected to a fixed loading condition.

In the second track, which is unsupervised learning, we aim to learn the unsupervised representation of microstructures. Along the line of this topic is the nonlinear dimensionality reduction framework for microstructure generation and reconstruction problems. However, the implication of unsupervised learning in microstructure is beyond microstructure generation and reconstruction and could have lasting impacts in community perception of process-structure-property linkages.

2. MICROSTRUCTURE DATASETS

In this project, as in many other DL applications for materials science, one of the computational bottlenecks is the lack of data to train. To mitigate this challenge, we utilize the large-scale computational resource from Sandia and use Big Compute to meet the demand of Big Data [5]. We consider five different microstructure datasets, where four of them consists of computational and synthetic microstructures, and one of them consists of experimental microstructures. For each synthetic dataset, except for the UHCSDB dataset, we generate 50,000 microstructures, where 40,000 are used for training and 10,000 are used for testing. Another potential candidate is the spinodal decomposition microstructure dataset via phase-field simulation, which was not considered in this study due to the prohibited computational cost to generate.

Section 2.1 describes the generation of synthetic microstructure images using Gaussian filter on a uniformly random field. Sections 2.2, 2.3, and 2.4 applies different SPPARKS simulations to generate microstructure images for welding with serpentine style, simple style, and grain growth. Section 2.5 discusses the possibility of applying on real experimental microstructures.

2.1. Synthetic microstructures

In this dataset, we adopt the microstructure generation workflow from Fernandez-Zelaia et al. [1], as shown in Figure 2-1, to generate synthetic microstructures with a slight modification. Figure 2-1

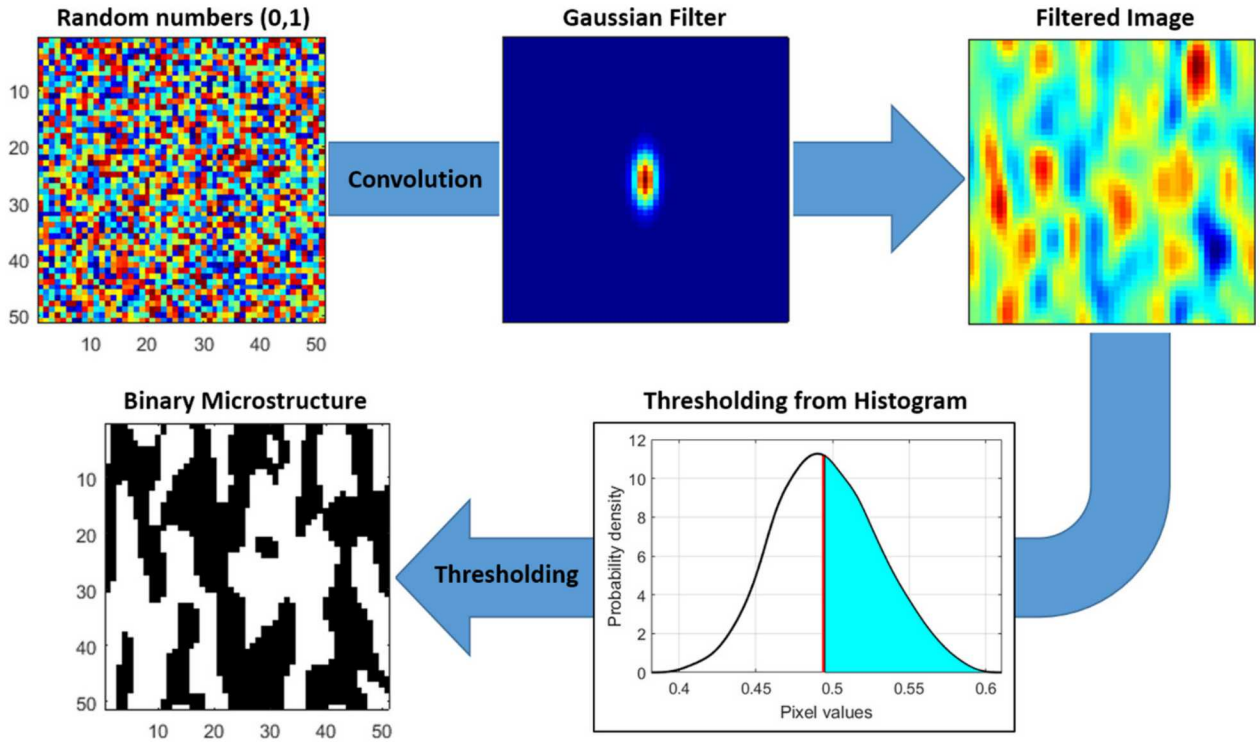


Figure 2-1 Synthetic microstructure generation. Photo courtesy of Fernandez-Zelaia et al. [1].

describes the workflow for generating synthetic microstructures. First, a uniformly random number is sampled at each pixel of an image, where dimension is a user-defined variable. Then, a Gaussian filter with periodic boundary condition, where the covariance matrix is also user-defined, is applied to filter the microstructure image. Finally, a binary/discrete microstructure is created by thresholding the "smooth" microstructure, where the threshold generally depends on the target volume of fraction. Because a grayscal image is more meaningful and contains more information than a binary image, we stop at the "smooth" microstructure image and does not apply any threshold for segmenting the microstructure. The dimension of the microstructure images is 128×128 for experimental purposes.

Figure 2-2 (top row) shows the smooth grayscale microstructure image before thresholding, where Figure 2-2 (bottom row) shows the binary microstructure after thresholding. It is easy to threshold the microstructure, i.e. from a smooth microstructure to a binary microstructure, but it is very challenging to go from a binary to a smooth microstructure.

2.2. SPPARKS/Potts additive small 2d

This dataset is generated by running the SPPARKS example 50,000 times using different seeds on a 100 sites $\times$ 100 sites computational domain. The processing parameters are fixed. The final output microstructure is then saved as a 512 pixel $\times$512 pixel .jpg image. Only the gray color version is retained, since the color version does not provide any additional information. Figure 2-3 shows a sample of 9 different microstructures with different seeds.

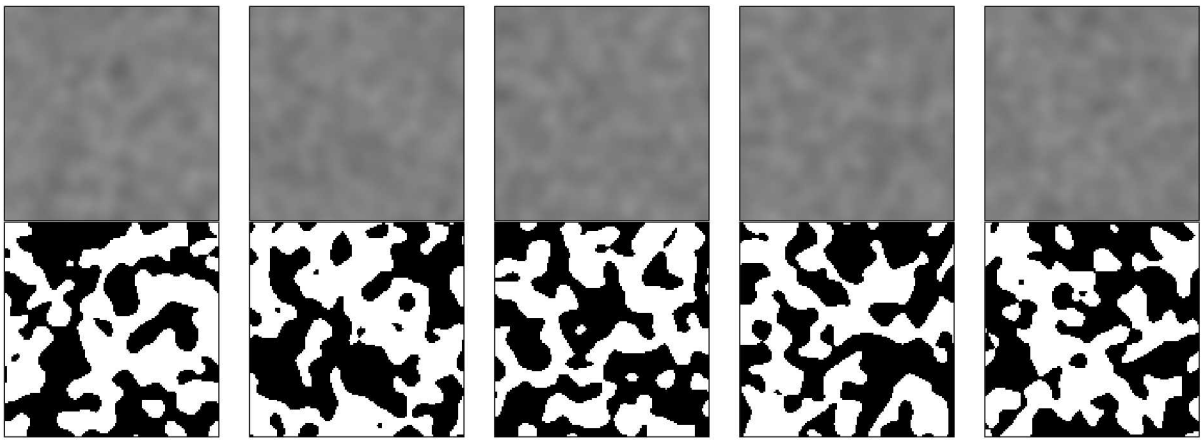


Figure 2-2 Synthetic random microstructure by uniform sampling and Gaussian filter. Top row: grayscal images. Bottom row: binary images.

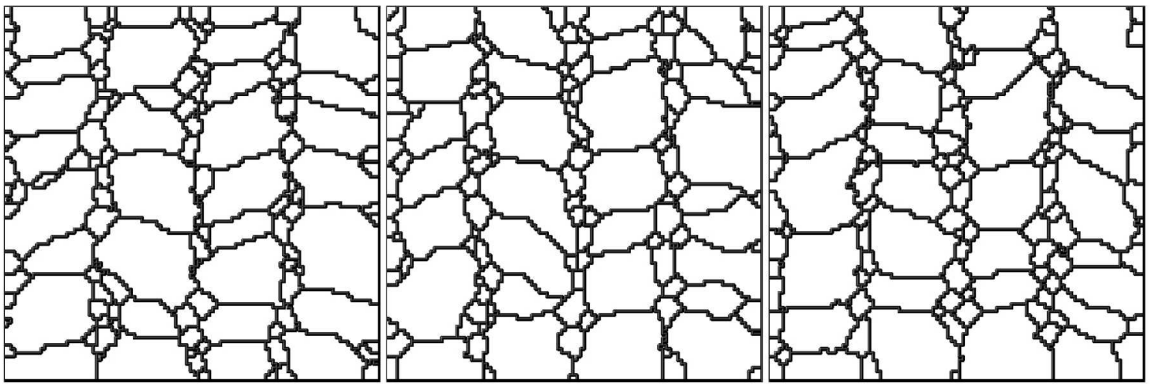
```
# SPPARKS potts/additive test file

seed         56789


app_style    potts/additive     1000 30 40 7 8      40 60 12 12 0.1

#  |-----------------------------------------------------------------------------
#  | nspins     = atoi(arg[1])
#  |-----------------------------------------------------------------------------
#  | nspins = atoi(arg[1]); #Number of spins
#  | spot_width = atoi(arg[2]); #Width of the melt pool
#  | melt_tail_length = atoi(arg[3]); #Length of tail from meltpool midpoint
#  | melt_depth = atoi(arg[4]); #How many lattice sites deep the melt pool is
#  | cap_height = atoi(arg[5]); #Height of the cap leading the meltpool
#  |-----------------------------------------------------------------------------
#  | HAZ = atoi(arg[6]); #Size of the HAZ surrounding the melt pool (must be larger than spot_width)
#  | tail_HAZ = atoi(arg[7]); #Length of hot zone behind meltpool (must be larger than melt_tail_length)
#  | depth_HAZ = atof(arg[8]); //Depth of the hot zone underneath the meltpool (must be larger than melt_depth)
#  | cap_HAZ = atoi(arg[8]); #Size of HAZ infront of the melt pool (must be larger than cap_height)
#  | exp_factor = atof(arg[9]); #Exponential parameter for mobility decay in haz M(d) = exp(-exp_factor * d)
#  |-----------------------------------------------------------------------------

#Define simulation domain and initialize site variables
#-----------------------------------------------------------------------------------
dimension    3
lattice      sc/26n 1.0
region       box block 0 100 0 100 0 1
region       transverse block 0 100 50 75 0 1
region       longitudinal block 50 75 0 100 0 1

boundary     n n n

create_box   box
create_sites     box
set      i1 range 1 1000
set      d1 value 0.0

#-----------------------------------------------------------------------------------

#Define an additive scan pattern using a combination of pass, transverse_pass, cartesian_layer, and pattern
#-----------------------------------------------------------------------------------

am_pass 1 dir X distance 100.0 speed 10
am_transverse_pass 1 distance 100.0 increment 25
am_cartesian_layer 1 start_position 0 0 pass_id 1 transverse_pass_id 1 serpentine 1
#am_cartesian_layer 2 start_position 0 0 pass_id 1 transverse_pass_id 1 serpentine 0
am_pattern 1 num_layers 1 layer_ids 1 z_start 0 z_increment 5

#Setup the solver type and parameters. Must use a "sweep" style solver
#-----------------------------------------------------------------------------------
sector       yes
sweep        random mask no
temperature  0.0
#-----------------------------------------------------------------------------------

#Specify output commands and styles.
#-----------------------------------------------------------------------------------
diag_style   energy
stats            1.0
#dump             1 text 5.0 dump.additive4.* id i1 d1
```

```
#If SPPARKS was not compiled with libjpeg, comment out the lines below.
# dump      top image 2 top.small2D.*.jpg site site crange 1 1000 drange 1 1 view 0.0 0.0 boundary site 1 shape cube box no 1 zoom 2 size 512 512 sdiam 1.05
# dump      mobility_top image 2 mobility_top.small2D.*.jpg d1 i1 view 0.0 0.0 shape cube size 512 512 sdiam 1.05 box no 1 zoom 2

## binary images only
dump            top image 2 top.small2D.*.jpg site site crange 1 1 drange 1 1 view 0.0 0.0 boundary site 1 shape cube box no 1 zoom 2 size 512 512 sdiam 1.05


dump_modify     top boundcolor black backcolor black pad 4
# dump_modify   mobility_top smap 0 1 cf 0.05 5 min blue 0.45 lightblue 0.65 yellow 0.75 orange max red

#dump transverse image 2 transverse.small2D.*.jpg site site crange 1 1000 center s 0.5 0.5 0.5 drange 1 1 view 90.0 -90.0 shape cube box no 1 zoom 1.5 size 512 512
#dump_modify transverse cwrap yes region transverse boundcolor black backcolor black pad 4

#dump longitudinal image 2 longitudinal.small2D.*.jpg site site crange 1 1000 drange 1 1 view 90.0 0.0 shape cube box no 1 zoom 1.5 size 512 512
#dump_modify longitudinal cwrap yes region longitudinal boundcolor black backcolor black pad 4
#-------------------------------------------------------------------------------

run             100
```

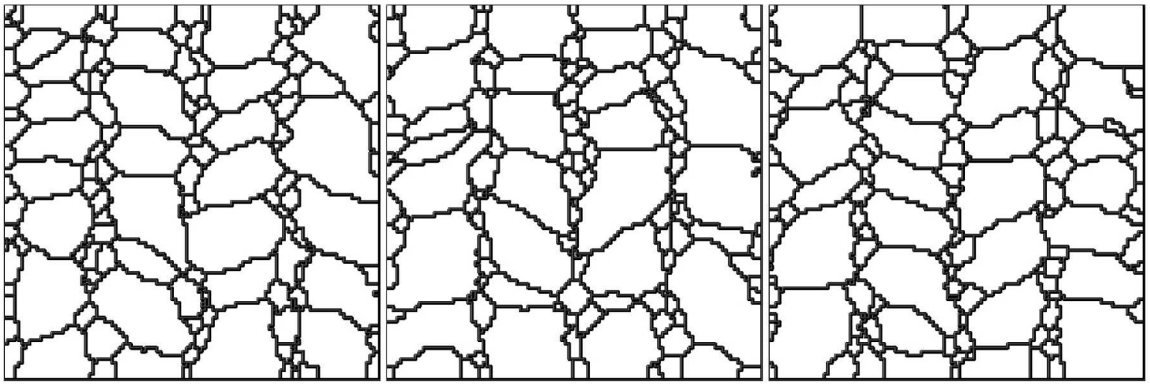
2.3. SPPARKS/Potts weld

In the same manner, the SPPARKS/Potts weld dataset is created. An initial computational domain of 200 sites $\times$ 500 sites is considered. A Potts/weld model is then applied with teardrop shape for the melting pool to create the microstructure. A final time-step is specified based on the velocity of the melting pool to ensure that the microstructure has been finalized. The final microstructure is then saved to a 1000 pixels $\times$ 400 pixels. To further reduce the size effect, we further crop the image along the welding axis from 1000 pixels to 600 pixels and take off 200 pixels on the left or the right of the microstructure image. Figure 2-4 shows a sample of 9 different welding microstructures using SPPARKS.

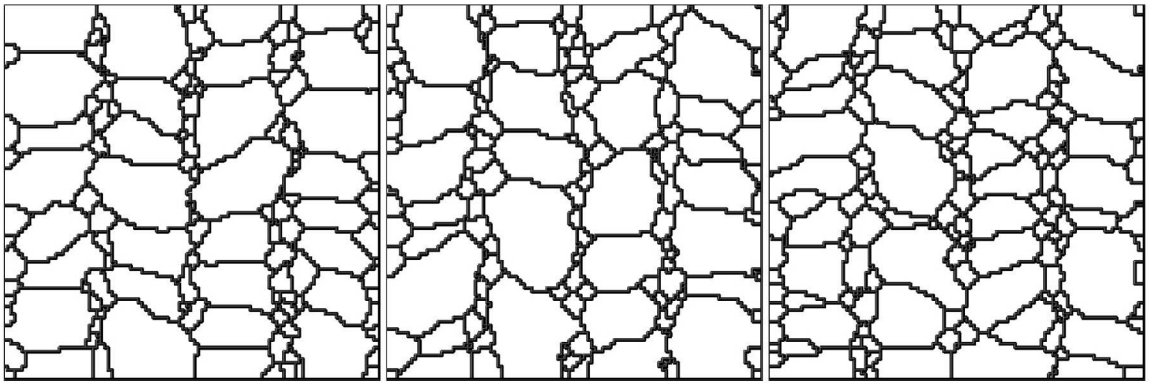
```
seed         123456

# app_potts_weld input parameters
#                                  POOL PARAMETERS
#                                 ---------------------------------
#          app_name   num_spins yp  alpha   beta   velocity haz
app_style  potts/weld 3500000   0   0.75    0.50     12.0   50.0
#                  width length
weld_shape_teardrop width 100.0 case III

# Run with 'pulse weld'
#     0<=pulse_amplitude<=1   2<frequency
# pulse    0.25                 32

dimension    3

# periodic x
# not periodic y
# not periodic z
# boundary p n n
boundary n n n

# NOTE: spk2vti writer does not currently support a lattice constant different from '1.0'
lattice      sc/26n 1.0
region       box block 0 200 0 500 0 1
create_box   box
create_sites     box

# Initialize grains
# read_sites site.init

# Run without grain initialization from 'potts_init'
# COMMENT out this line if using 'read_sites' above
set site range 1 3500000

sweep        raster
sector yes

diag_style       energy

# Simulation temperature
temperature  0.25
```



(a) Sample 1 (b) Sample 2 (c) Sample 3



(d) Sample 4 (e) Sample 5 (f) Sample 6

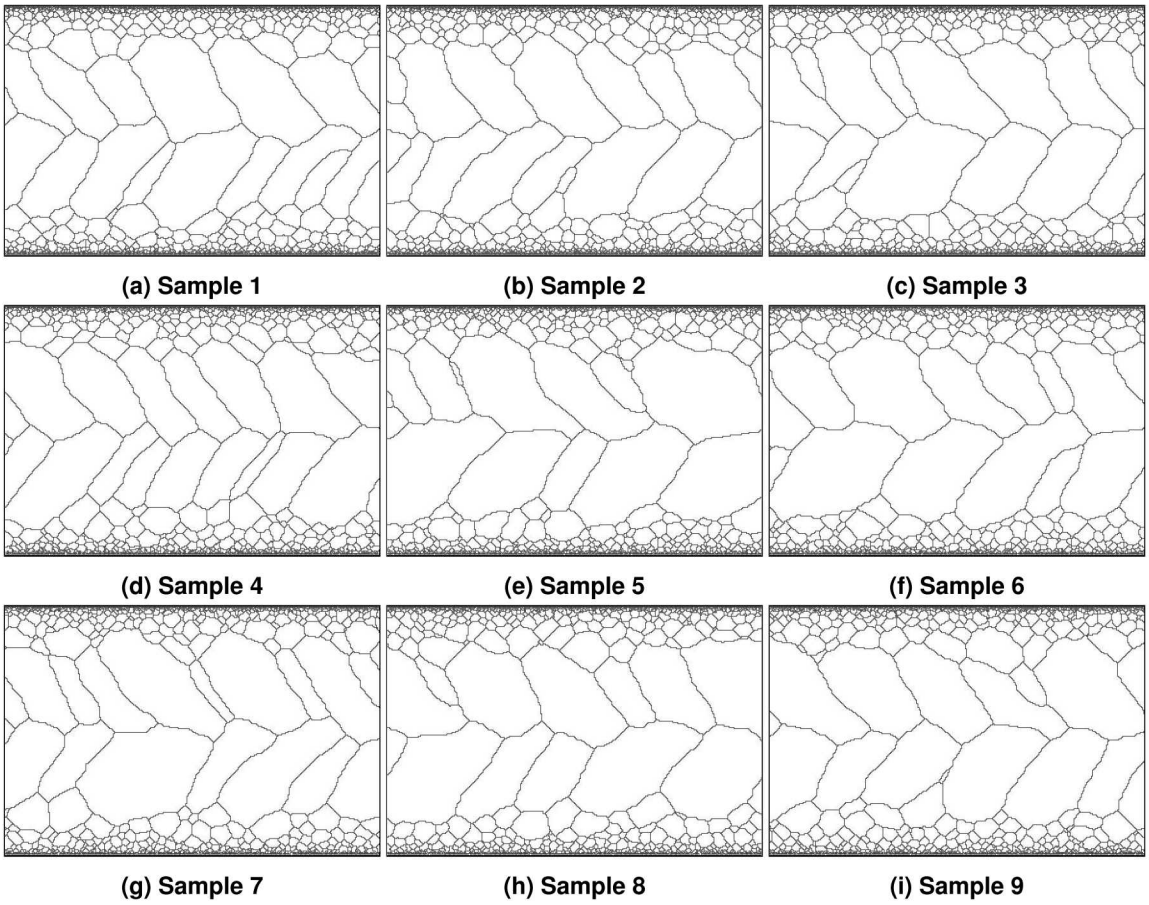


(g) Sample 7 (h) Sample 8 (i) Sample 9

Figure 2-3 SPPARKS Potts/additive microstructures.

```
stats            1.0

# # Write 'dump' files; see 'dump' documentation
# dump             1 text 52.0 steady_weld.dump
# dump_modify      1 delay 52.0
# diag_style       cluster stats no delay 52.0 delt 52.0 filename steady_weld.cluster

# Write 'image'; see 'dump' documentation
#    images written every 2 seconds
# dump      top image 2 top.*.jpg site site crange 1 3500000 drange 1 1 view 0.0 0.0 boundary site 1 shape cube box no 1 zoom 2.0 size 1024 1024 sdiam 1.05
dump           teardrop image 2 teardrop_*.jpg site site crange 1 1 drange 1 1 view 0.0 0.0 boundary site 1 shape cube box no 1 zoom 5.0 size 1000 400 view 0.0 0.0 sdiam 1.05
# dump_modify   top cwrap yes boundcolor black backcolor black pad 4
dump_modify     teardrop boundcolor black backcolor black pad 0

# dump      bottom image 2 bottom.*.jpg site site crange 1 3500000 drange 1 1 view 180.0 0.0 boundary site 1 shape cube box no 1 zoom 2.0 size 1024 1024 sdiam 1.05
# dump_modify   bottom cwrap yes boundcolor black backcolor black pad 4

run             52.0
```

```
################ cropMs.py
import numpy as np
import skimage
import skimage.io as io

img = skimage.io.imread('teardrop_26.jpg')
# img.shape

crop = img[:, 200:800, :]
io.imsave('cropTeardrop_26.jpg', crop)
```



(a) Sample 1 (b) Sample 2 (c) Sample 3

(d) Sample 4 (e) Sample 5 (f) Sample 6

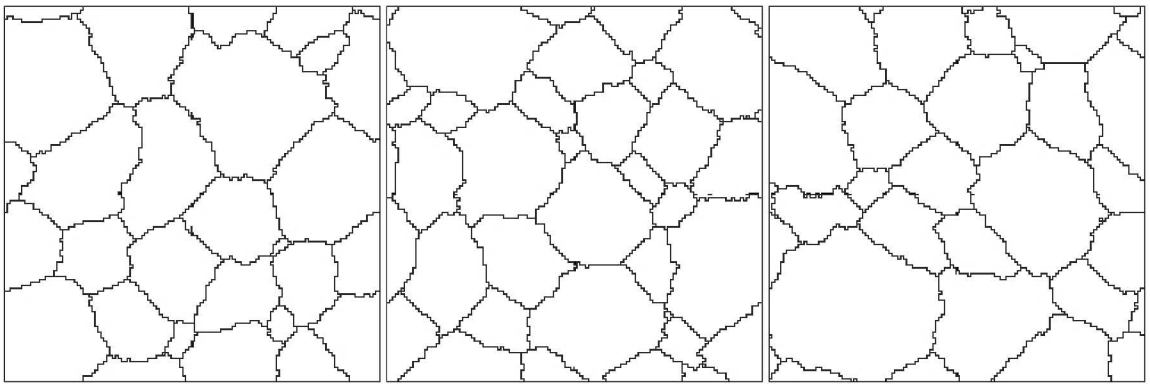
(g) Sample 7 (h) Sample 8 (i) Sample 9

Figure 2-4 SPPARKS Potts/weld additive manufacturing microstructures.

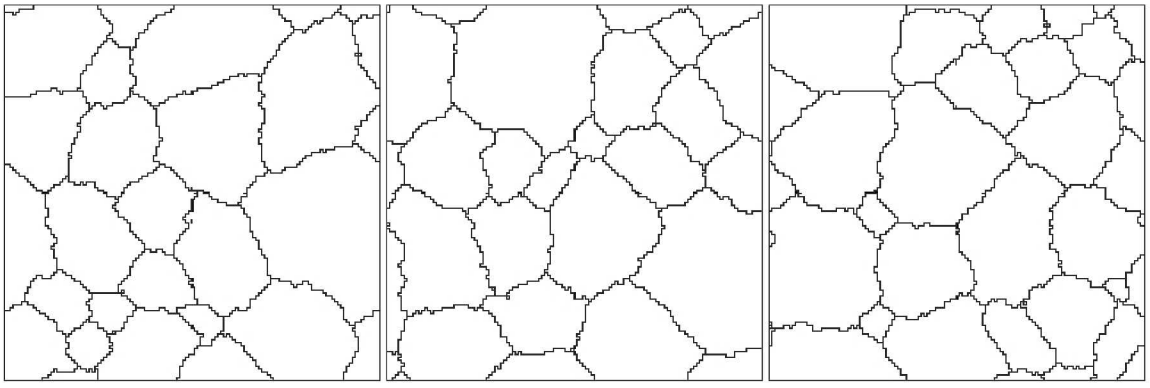
2.4. SPPARKS/Grain growth

The SPPARKS/Potts grain growth simulation is used to generate microstructures. A kT of 0.85 is used on a 128 sites $\times$ 128 sites computational domain. The grain growth simulation is performed for 1500 Monte Carlo step and the final microstructure is obtained. Figure 2-5 shows 9 different microstructures obtained from the SPPARKS grain growth simulation. The final size of microstructure image in this dataset is 256 pixels $\times$ 256 pixels.

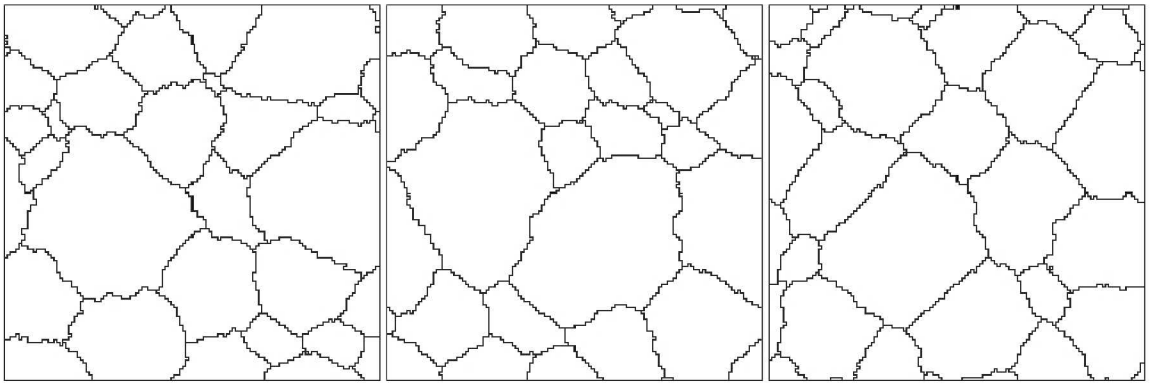
```
# SPPARKS 'potts' model for modeling 'grain' growth
# let N=128, 256
# let Q=2048
# kT="0.85"
# spk_flame.gnu -var kT $kT -var N $N -var Q $Q < grain_growth.in

variable    N equal 128
variable    Q equal 2048
variable    kT equal 0.85

seed         56897

# Use 'Potts' app to simulate grain growth;
# Site 'spins' can assume values of its nearest neighbors.
app_style    potts/neighonly ${Q}

# 2d lattice; Each site has 8 neighbors
dimension    2
lattice      sq/8n 1.0

# Define lattice extent called 'square';
region       square block 0 ${N} 0 ${N} -1.0 1.0

# Define 'axis' aligned simulation domain on 'square'
create_box   square

# Creates sites on lattice defined above;
# Also creates neighborhood list for each site
create_sites     box

# Initializes sites randomly with values between 1 and {Q}
set site range 1 ${Q}

# Simulation temperature; Units of energy
temperature  ${kT}

# KMC solver
solve_style      tree
# sectors are required to run 'tree' kmc in parallel
sector yes

# Diagnostic style energy computes the total energy for all sites
diag_style       energy

# Print diagnostic information to screen 50 steps
stats            50.0

# Write snapshot of site values
#   to 'grain_growth.dump' every 50 steps
# dump          1 text  50.0 grain_growth.dump
# Write 'cluster/grain' diagnostics
diag_style       cluster stats no delt 50.0 filename grain_growth.cluster

# Write 'image'; see 'dump' documentation
variable        zoomFactor equal 2.00
variable        imageDim equal ${N}*${zoomFactor} # or 2*${N}*${zoomFactor}

# dump          grain_growth_image_Color image 1 grain_growth_Color.*.jpg site site crange 1 ${Q} drange 1 1 view 0.0 0.0 boundary site 1 shape cube box no 1 zoom 1.95 size ${im
# dump_modify   grain_growth_image_Color boundcolor black backcolor white pad 1

dump            grain_growth_image_Bnw image 1 grain_growth_Bnw.*.jpg site site crange 1 1 drange 1 1 view 0.0 0.0 boundary site 1 shape cube box no 1 zoom ${zoomFactor} size ${
dump_modify     grain_growth_image_Bnw boundcolor black backcolor white pad 0

# Run for 1400 *spparks* steps
run              1500.0
```



(a) Sample 1 (b) Sample 2 (c) Sample 3



(d) Sample 4 (e) Sample 5 (f) Sample 6

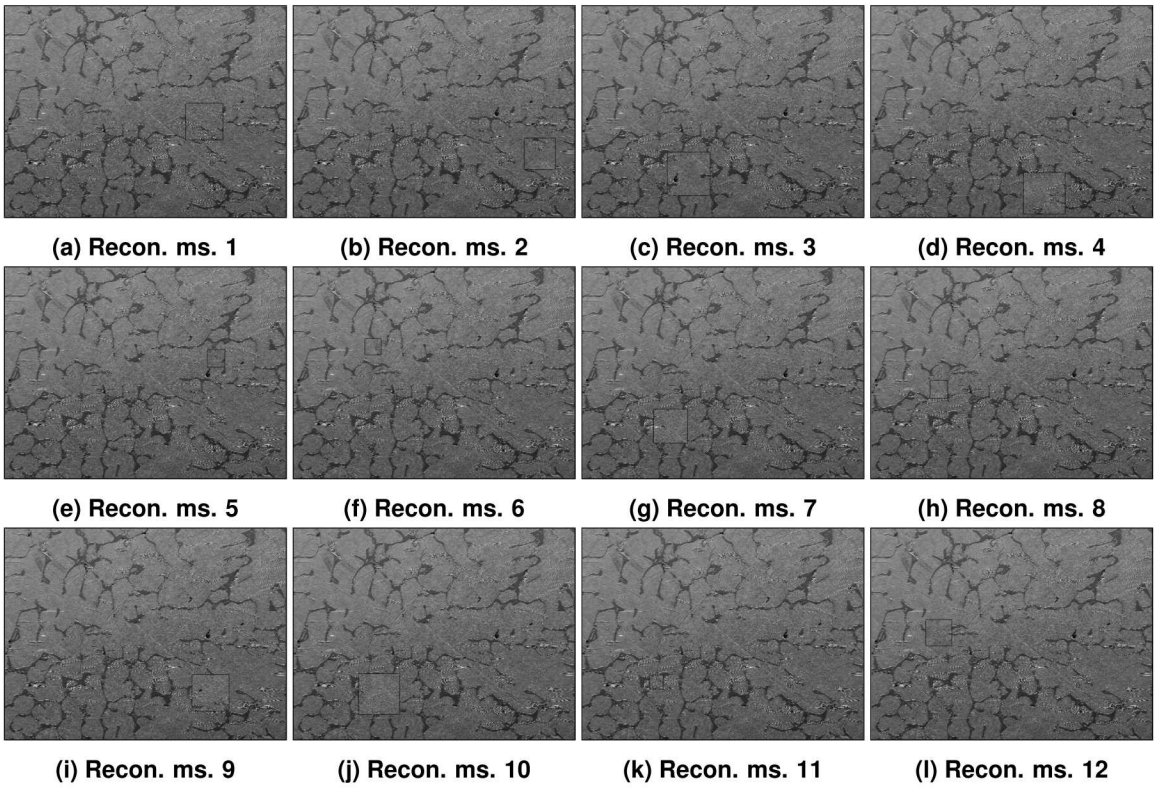


(g) Sample 7 (h) Sample 8 (i) Sample 9

Figure 2-5 SPPARKS Potts/grain growth microstructures.

2.5. UltraHigh Carbon Steel DataBase (UHCSDB) with image-inpainting

The dataset is created using the novel image inpainting method proposed by Tran and Tran [6] recently applying on the high-fidelity microstructure UHCSDB dataset. Figure 2-6 shows 12 variants of the same original microstructure (micrograph #13). Potentially, the image inpainting technique can help mitigate the demand of Big Data to train DL architectures for microstructure.



(a) Recon. ms. 1 (b) Recon. ms. 2 (c) Recon. ms. 3 (d) Recon. ms. 4

(e) Recon. ms. 5 (f) Recon. ms. 6 (g) Recon. ms. 7 (h) Recon. ms. 8

(i) Recon. ms. 9 (j) Recon. ms. 10 (k) Recon. ms. 11 (l) Recon. ms. 12

Figure 2-6 Inpainting UHCSDB microstructures.

2.6. Localization datasets

Based on our established collaboration with Prof. Surya Kalidindi and Dr. Yuksel Yabansu, we obtain the localization dataset for two-phase composite materials which has been used to validate the Materials Knowledge System (MKS) framework. Other datasets are available on MATIN platform [7] at Georgia Tech. The dataset focused in this study was initially proposed in [8] in developing and validating the MKS framework, and subsequently expanded over years [9].

3. LITERATURE REVIEWS

Iyer et al. [10] and Chun et al. [11] utilized GAN to generate and reconstruct microstructure. Fokina et al. [12] employed StyleGAN to generate and reconstruct microstructure. From the creation of

GAN by Goodfellow et al. [13], further improvements have been made, chronologically as Radford et al. [14], Liu et al. [15], Karras et al. [16], and Karras et al [17]. For example, Bostanabad [18] adopted VGG19 [19] to reconstruct 3D microstructure from 2D images using transfer learning. Iyer et at. [10] employed an auxiliary classifier Wasserstein GAN with gradient penalty to generate microstructure from UHCSDB, which is the same dataset considered in this work. Singh et al. [20] used Wasserstein GAN to generate and reconstruct microstructure with binary phases. DeCost et al. [21] applied VGG16 and t-SNE [22, 23] to visualize microstructure on their latent manifold space. DeCost et al. [24] applied a pre-trained VGG16 [19] for deep semantic segmentation in the same UHCSDB dataset. Ling et al. [25] also used VGG16 to extract features for SEM images between different datasets in the hope of generalization and interpretation. Li et al. [26] employed an auto-encoder (AE) approach to generate microstructures. Chun et al. [11] employed GAN to generate microstructures and showed that GAN is able to generate better quality images compared to AE, which is a well-known problem in computer vision. Mosser et al. [27] proposed a GAN to generate microstructure. Cang et al. [28, 29] employed deep belief network in reconstructing binary microstructure. Bostanabad et al. [30] proposed a tree-based ML technique for 2D stochastic microstructure reconstruction based on classification trees.

Zichenko [31] proposed an isotropic algorithm for random close packing of equi-sized spheres with periodic boundary conditions. Groeber et al. [32, 33, 34] proposed an automatic statistical framework to characterize [33] and to create statistically equivalent synthetic microstructures [34]. Fullwood et al. [35, 36] proposed a phase-recovery algorithm based on two-point correlation statistics to reconstruct the microstructure. Latief et al. [37] suggested a stochastic geometrical modeling approach to generate a μ-CT images of Fontainebleau sandstone. Staraselski et al. [38] demonstrated the application of two-point correlation function in constructing 3D representative volume element. Feng et al. [39] proposed a stochastic microstructure reconstruction for two-phase composite materials based on nonlinear transformation of Gaussian random fields that matches the marginal probability distribution function and the two-point correlation function. Chen et al. [40] employed simulated annealing method to reconstruct 3D multiphase microstructure and demonstrated with 2D and 3D reconstruction with three-phase sandstone. Xu et al. [41, 42] proposed a descriptor-based methodology using multiple microstructure descriptors as evaluation criteria to reconstruct 3D microstructure. Chen et al. [43] proposed a multiscale computational scheme to stochastically reconstruct the 3D heterogeneous polycrystalline microstructure from a single 2D electron back-scattered diffraction (EBSD) micrograph. Tran et al. [6] proposed a non-local patch-based image inpainting to reconstruct microstructures at the experimental level. Li et al. [44, 45] conducted a comparison study on the effects of multiple objectives in the microstructure reconstruction problem.

4. SOLVING LOCALIZATION PROBLEM VIA DEEP LEARNING TECHNIQUES

In this section, we utilize the supervised DL technique to learn the map from (micro)structure to the (localized) property, which is the material response, i.e. strain, under the imposed loading boundary condition. Before getting to the details, we note that brute-force application for directly

solving localization without subsampling does not work, because it could result in a computationally intractable DNN to learn. We follow the approach in Yang et al. [46] due to this reason.

Elastic prediction using MKS [47, 48] and its extension for plastic extension has been proposed by Yabansu et al. [47, 48] de Oca Zapiain et al [49], respectively. The main idea is to construct a reduced-order model based on two-point statistics and to carefully calibrate the coefficients. The MKS is then used to predict the local materials response, such as strain field. Liu et al. [50, 51] developed a physics-based microstructure descriptors approach to parameterize microstructures as inputs and constructed the map using regression trees and support vector regressors. Recently, DL techniques have been proposed to solve the localization problem [46] and the homogenization problem [52, 53] subsequently. Generally speaking, the homogenization problem is much easier to solve because the quantity of interest (QoI) is single-output, as opposed to multi-output in the localization problem.

In this section, we implement and examine a variety of DL architectures in PyTorch that are capable of solving localization problems using the same datasets. 6840 microstructures are used as the training dataset, 1710 microstructures are used as the testing dataset, which constitute a dataset of 8550 microstructure with 80/20 split.

```
% linkTesting.sh
#!/bin/
# M_gaussian_7_7_7_vf_25_75_set2.Idx0.npy # _set2_ range from .Idx0. to .Idx99.
# M_gaussian_7_7_7_vf_25_75.Idx0.npy # no annotation (_set1_) range from .Idx0. to .Idx49.
# train: 80%
# test:  20%
_filterName="_SplineFilterOrder2"
# loop over (40,49) out of (0,49) for _set2_
for i in $(seq 40 49); do
        for ii in 1 3 5 7; do
                for jj in 1 3 5 7; do
                        for kk in 1 3 5 7; do
                                ln -sf ../microstructures/M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
                                ln -sf ../responses/Strain.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
                                ln -sf ../responses/Stress.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
                                ln -sf ../_SplineFilterOrder2_Microstructures/_SplineFilterOrder2_M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
                        done
                done
        done
done

# remove broken links
find -L . -name . -o -type d -prune -o -type l -exec rm {} +
```

```
% linkTraining.sh
#!/bin/
# M_gaussian_7_7_7_vf_25_75_set2.Idx0.npy # _set2_ range from .Idx0. to .Idx99.
# M_gaussian_7_7_7_vf_25_75.Idx0.npy # no annotation (_set1_) range from .Idx0. to .Idx49.
# train: 80%
# test:  20%
_filterName="_SplineFilterOrder2"
# loop over (0,39) out of (0,49) for _set2_
for i in $(seq 0 39); do
  for ii in 1 3 5 7; do
    for jj in 1 3 5 7; do
      for kk in 1 3 5 7; do
        ln -sf ../microstructures/M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
        ln -sf ../responses/Strain.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
        ln -sf ../responses/Stress.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
                                ln -sf ../rescaled_responses/RescaledStrain.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
                                ln -sf ../rescaled_responses/RescaledStress.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
        ln -sf ../_SplineFilterOrder2_Microstructures/_SplineFilterOrder2_M_gaussian_${ii}_${jj}_${kk}_vf_25_75.Idx${i}.npy
      done
    done
  done
done
# loop over (0,79) out of (0,99) for _set2_
for i in $(seq 0 79); do
  for ii in 1 3 5 7; do
    for jj in 1 3 5 7; do
      for kk in 1 3 5 7; do
        ln -sf ../microstructures/M_gaussian_${ii}_${jj}_${kk}_vf_25_75_set2.Idx${i}.npy
        ln -sf ../responses/Strain.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75_set2.Idx${i}.npy
        ln -sf ../responses/Stress.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75_set2.Idx${i}.npy
                                ln -sf ../rescaled_responses/RescaledStrain.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75_set2.Idx${i}.npy
```

```
                            ln -sf ../rescaled_responses/RescaledStress.Resp_M_gaussian_${ii}_${jj}_${kk}_vf_25_75_set2.Idx${i}.npy
      ln -sf ../_SplineFilterOrder2_Microstructures/_SplineFilterOrder2_M_gaussian_${ii}_${jj}_${kk}_vf_25_75_set2.Idx${i}.npy
    done
  done
 done
done
# remove broken links
find -L . -name . -o -type d -prune -o -type l -exec rm {} +
```

4.1. Microstructure filter

The role of the microstructure filter is to convert the representation of the microstructure from discrete representation (which is binary in this case) to continuous representation. It is common for engineers and materials scientists to encode the type of materials under discrete representation, e.g. hard phase as 0 and soft phase as 1. The microstructure filter effectively converts the approximation problem of DL from $f : 2^{d^3} \rightarrow \mathcal{R}^{d^3}, f(\boldsymbol{m}) = \boldsymbol{r}$ to $f : \mathcal{R}^{d^3} \rightarrow \mathcal{R}^{d^3}, f(\boldsymbol{m}) = \boldsymbol{r}$, where $\boldsymbol{m}$ is the microstructure and $\boldsymbol{r}$ is the microstructure response. The first problem is NP-hard, as it is analogous to combinatorial optimization problem, whereas the second problem is P-hard.

4.2. Normalization

```
% /ascldap/users/anhtran/scratch/dataset/matin/mveLocalization/high_contrast_elastic/subsamples/_SplineFilterOrder2/train
import numpy as np
fileList = np.loadtxt('msFileList.txt', dtype=str)
n = len(fileList)
meanArr = []
stdArr = []

prefix = '' # '', 'Stress.Resp_', 'Strain.Resp_'

for i in range(n):
  ms = np.load(prefix + fileList[i])
  meanArr += [ms.mean()]
  stdArr += [ms.std()]
  print('done %d' % i)

print(np.mean(np.array(stdArr))) # based on the law of large number
print(np.mean(np.array(meanArr))) # based on the law of large number
```

After filtering the microstructures, we estimate the mean and standard deviation of QoIs as follows.

- Ms: $\mu = 0.5506806080114989; \sigma = 0.4831280930966276$.
- Stress: $\mu = 0.038860694801369854; \sigma = 0.03840891767246447$.
- Strain: $\mu = 0.0009999999999876312; \sigma = 0.0008990865991095956$.
- BlurredMs: $\mu = 0.814312086506682; \sigma = 0.5549198340004976$.

Then, the inputs SVE and outputs SVE responses are normalized with their corresponding means and standard deviations, respectively.

```
% rescaleStrainFile.py
import numpy as np
fileList = np.loadtxt('strainFileList.txt', dtype=str)
n = len(fileList)
for i in range(n):
  strain = np.load(fileList[i])
  rescaledStrain = (strain - 0.0009999999999876312) / 0.0008990865991095956
  fileName = 'Rescaled' + fileList[i]
  np.save(fileName, rescaledStrain, allow_pickle=True, fix_imports=True)
  print('done %d' % i)
```

The script to visualize microstructures and responses (ε, σ) in 3D is documented as below.

```
import numpy as np
import matplotlib.pyplot as plt
import os, glob
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.axes_grid1.colorbar import colorbar

currentPath = os.getcwd()
fileList = glob.glob(currentPath + '/microstructures/*npy')
# i = np.random.randint(low=0, high=len(fileList))

for i in range(len(fileList)):
  fileName = fileList[i]
  fileName = fileName.split('/')[-1].replace('.npy', '')

  # fileName = 'M_gaussian_5_7_7_vf_25_75_set2.Idx10'
  data = np.load('microstructures/' + fileName + '.npy')
  data2 = np.load('responses/Stress.Resp_' + fileName + '.npy')

  ### plot

  plt.close('all')

  fig = plt.figure()
  # ax1 = fig.gca(projection='3d')
  ax1 = fig.add_subplot(1,2,1, projection='3d')

  cmap = plt.get_cmap("gray")
  norm = plt.Normalize(data.min(), 2 * data.max())
  ax1.voxels(np.ones_like(data), facecolors=cmap(norm(data)))
  ax1.set_title('microstructures')
  ax1.set_aspect('equal')

  # fig2 = plt.figure()
  # ax2 = fig2.gca(projection='3d')
  ax2 = fig.add_subplot(1,2,2, projection='3d')

  cmap = plt.get_cmap("jet")
  norm = plt.Normalize(data2.min(), 0.25 * data2.max())
  ax2.voxels(np.ones_like(data2), facecolors=cmap(norm(data2)))
  ax2.set_aspect('equal')
  ax2.set_title('Stress')
  # fig2.colorbar(ax2)

  plt.savefig('visualization.%s.png' % fileName, quality=95)
  plt.clf()
  print('done %d' % i)
# plt.show()
```

4.3. CPNet architecture

```
Total number of params: 902,183
Total number of trainable params: 902,183
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
        Conv3d-1            [-1, 8, 11, 11, 11]             224
   BatchNorm3d-2            [-1, 8, 11, 11, 11]              16
          ReLU-3            [-1, 8, 11, 11, 11]               0
     MaxPool3d-4            [-1, 8, 11, 11, 11]               0
        Conv3d-5              [-1, 16, 6, 6, 6]           3,472
   BatchNorm3d-6              [-1, 16, 6, 6, 6]              32
          ReLU-7              [-1, 16, 6, 6, 6]               0
     MaxPool3d-8              [-1, 16, 6, 6, 6]               0
        Conv3d-9              [-1, 32, 6, 6, 6]          13,856
   BatchNorm3d-10             [-1, 32, 6, 6, 6]              64
          ReLU-11             [-1, 32, 6, 6, 6]               0
```

```
      MaxPool3d-12              [-1, 32, 5, 5, 5]               0
         Conv3d-13              [-1, 64, 3, 3, 3]          55,360
    BatchNorm3d-14              [-1, 64, 3, 3, 3]             128
           ReLU-15              [-1, 64, 3, 3, 3]               0
      MaxPool3d-16              [-1, 64, 3, 3, 3]               0
         Linear-17                     [-1, 343]         593,047
           ReLU-18                     [-1, 343]               0
         Linear-19                     [-1, 343]         117,992
           ReLU-20                     [-1, 343]               0
         Linear-21                     [-1, 343]         117,992
================================================================
Total number of params: 902,183
Total number of trainable params: 902,183
Non-trainable params: 0
```

4.4. Prediction benchmark

One of the computational trade-offs in CPNet is the number of hyper-parameters included in CPNet versus the speed for forward prediction. In the forward prediction mode, a smaller scale materials response of SVE strideGap×strideGap×strideGap is made as an output of the CPNet. This procedure sweeps for the entire the SVE. Larger strideGap is associated with a dense fully convoluted layer after feature extraction, thus significantly increasing the number of hyper-parameters of CPNet. Smaller strideGap reduces the number of hyper-parameters, but increases the computational cost of the forward prediction, as demonstrated in Figure 4-1.

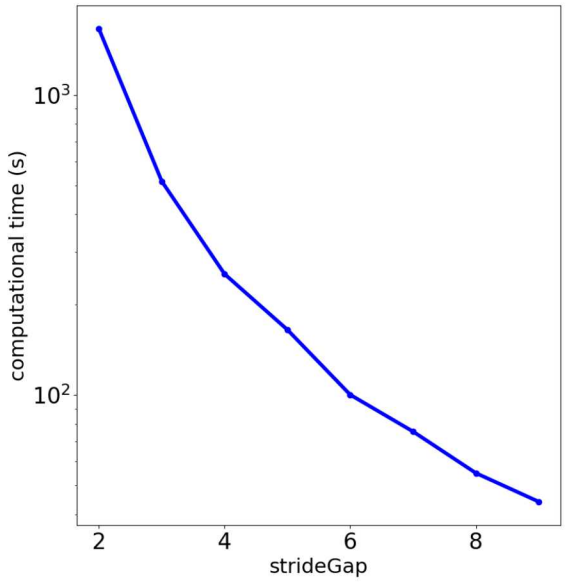


Figure 4-1 CPNet scalability wrt strideGap in forward prediction for 1 SVE.

Beside strideGap, batchSize also has a big impact on the training performance. In particular, one expects:

- 1 epoch for batchSize = 16,
- 3-4 epochs for batchSize = 32,
- 7-8 epochs for batchSize = 64.

Figure 4-2 and Figure 4-3 present the prediction with strideGap of 7 and 5, respectively.

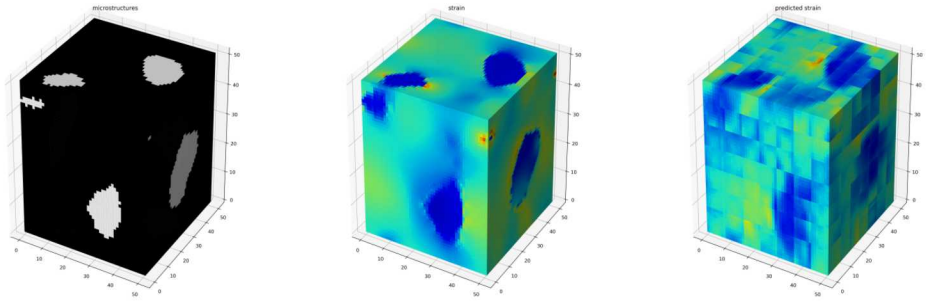


Figure 4-2 CPNet - strideGap = 7 for testing.

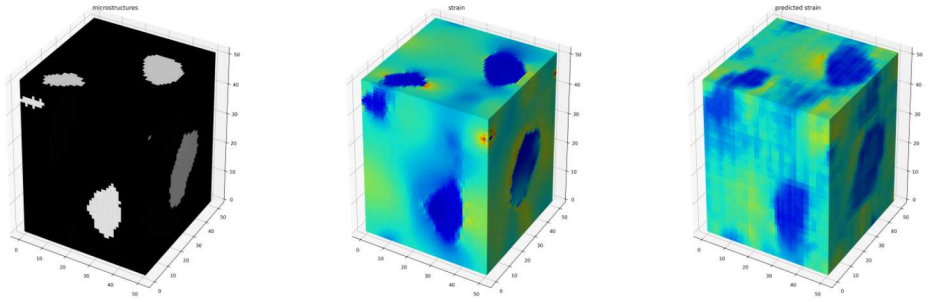


Figure 4-3 CPNet - strideGap = 5 for testing.

Figures 4-4 to 4-7 show the performance of CPNet with various microstructures. The left column shows the microstructures, which are the input of CPNet. The middle column shows the prediction of localized strain. The right column shows the true localized strain by FEM.

4.5. Quantitative comparison

For a SVE of size $51 \times 51 \times 51$, which has 132,651 data points to compare and compute the R^2 coefficients. However, for a testing dataset of 1720 SVEs, which totals up to $0.22 \cdot 10^9$ data

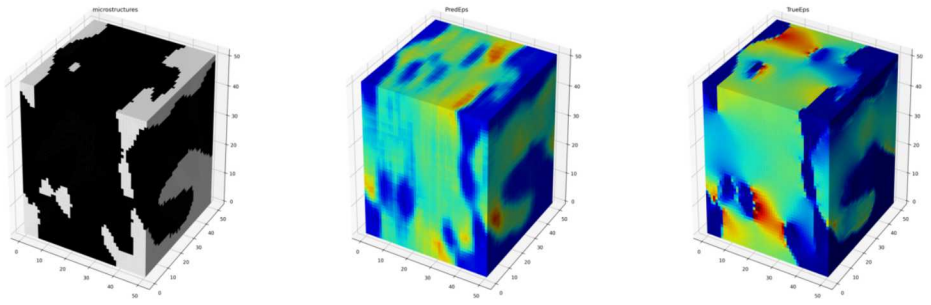


Figure 4-4 M_gaussian_3_7_5_vf_25_75_set2.Idx99

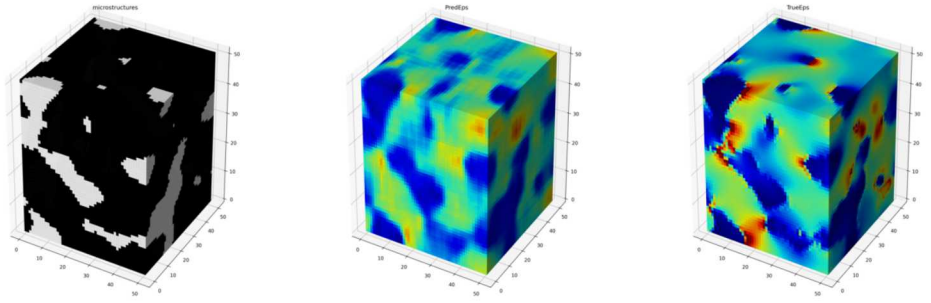


Figure 4-5 M_gaussian_5_5_3_vf_25_75_set2.Idx99

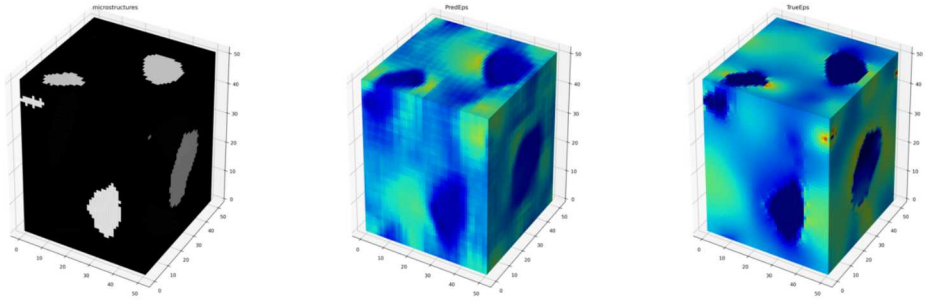


Figure 4-6 M_gaussian_7_7_7_vf_25_75_set2.Idx99

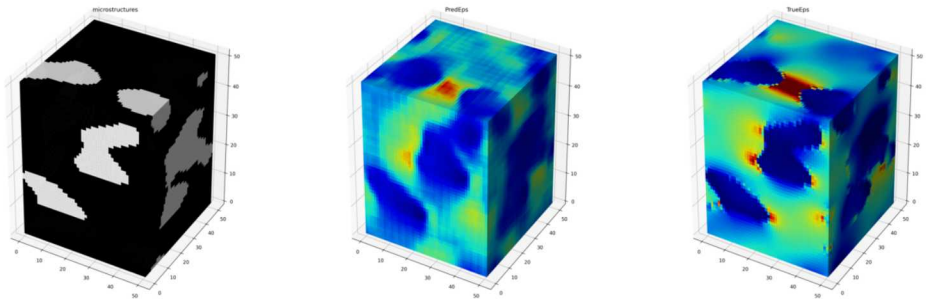


Figure 4-7 M_gaussian_7_5_3_vf_25_75_set2.Idx99

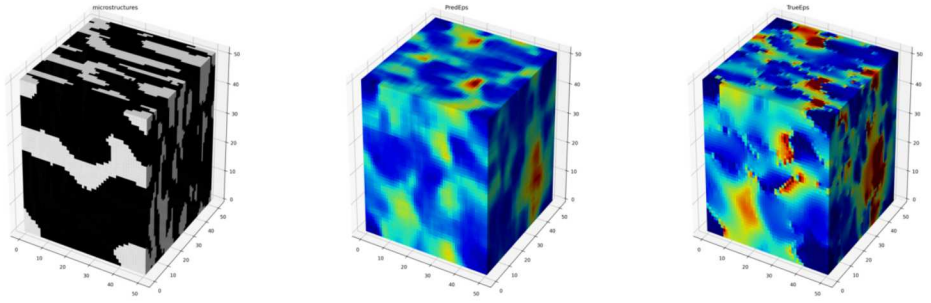


Figure 4-8 M_gaussian_5_1_3_vf_25_75.Idx42

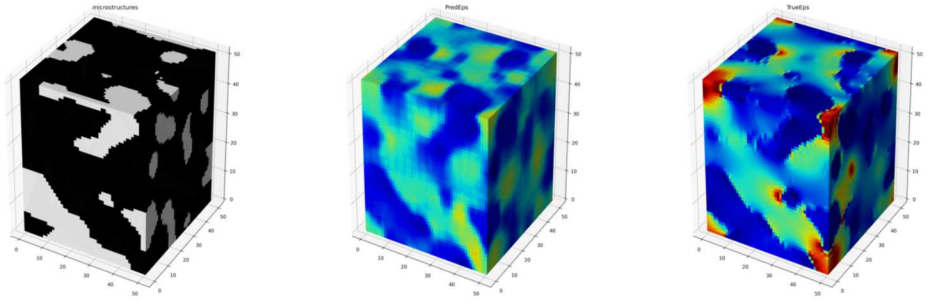


Figure 4-9 M_gaussian_5_3_3_vf_25_75_set2.Idx99

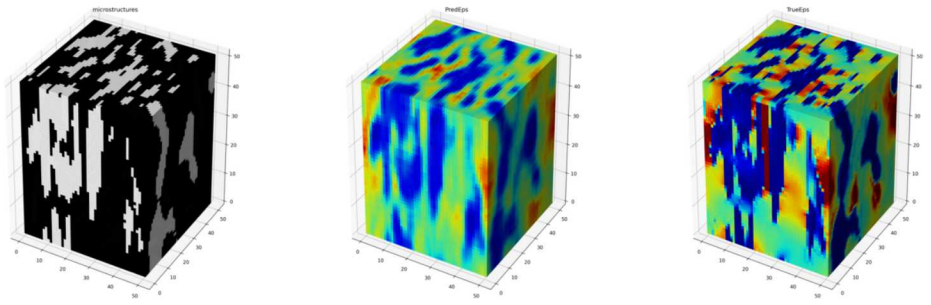


Figure 4-10 M_gaussian_1_3_5_vf_25_75_set2.Idx94

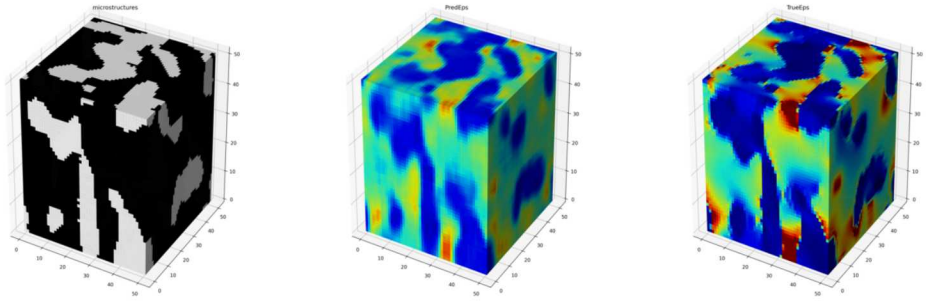


Figure 4-11 M_gaussian_3_3_5_vf_25_75_Idx42

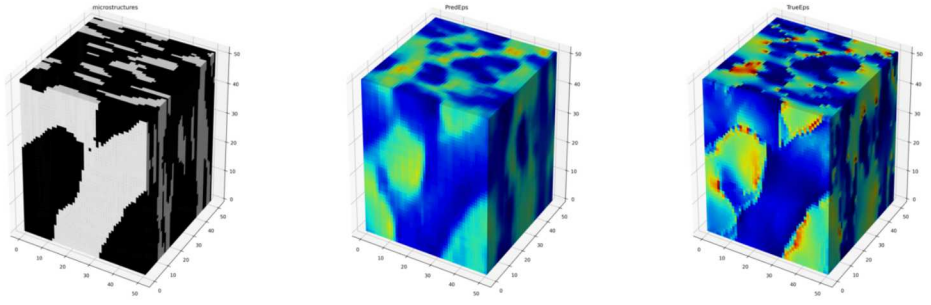


Figure 4-12 M_gaussian_5_1_7_vf_25_75_set2.Idx85.npy.png

points, there is a computational issue if all the data points are included. Thus, we have to resort to some approximation technique for assessing the R^2 coefficients. For the sake of simplicity, we approximate this coefficient by Monte Carlo methods, where the number of SVEs considered is much less than the number of SVEs included in the testing dataset.

Given n points to compare between the prediction and ground-truth, the R^2 coefficient is the square of Pearson's correlation coefficient, which is computed by

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{1}$$

where n is the sample size, x_i and y_i are the prediction and ground-truth values, respectively, $\bar{x}$ and $\bar{y}$ denote sample means. It can be proved mathematically that the R^2 coefficient does not change under affine transformation. Therefore, we will compute this R^2 coefficient without scaling the prediction and ground-truth values altered during the pre-process of DL. In this implementation, CPNet achieves about $0.65 \leq R^2 \leq 0.70$. One possible explanation is that the dataset is insufficient to train a large CNN.

5. AUTOENCODER FOR SYNTHETIC 2D SMOOTH MICROSTRUCTURES

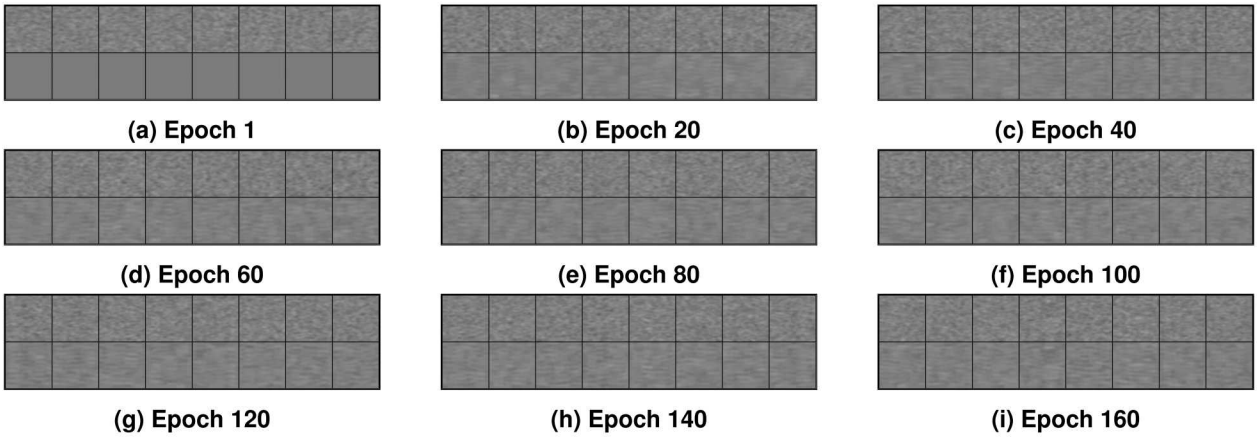
The verbal description of hyper-parameters and architecture is described as follows via torchsummary,

```
Total number of params: 8801
Total number of trainable params: 8801
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 43, 43]             160
              ReLU-2          [-1, 16, 43, 43]               0
         MaxPool2d-3          [-1, 16, 21, 21]               0
            Conv2d-4           [-1, 8, 11, 11]           1,160
              ReLU-5           [-1, 8, 11, 11]               0
         MaxPool2d-6           [-1, 8, 10, 10]               0
   ConvTranspose2d-7          [-1, 16, 21, 21]           1,168
              ReLU-8          [-1, 16, 21, 21]               0
   ConvTranspose2d-9           [-1, 8, 65, 65]           6,280
             ReLU-10           [-1, 8, 65, 65]               0
  ConvTranspose2d-11         [-1, 1, 128, 128]              33
            Tanh-12         [-1, 1, 128, 128]               0
================================================================
Total params: 8,801
Trainable params: 8,801
Non-trainable params: 0
```

where the detailed architecture is shown in Figure 5-1. Figure 5-1 shows the architecture of the AE used. Figure 5-3 presents 64 different samples of microstructures using the decoder of the AE. It is shown that the AE is capable of producing synthetic 2D microstructures to a fair level of details. However, it is also noted that the quality of images are not as good as the original ones.

Figure 5-1 AE DL architecture for synthetic 2D microstructures.

Figure 5-2 shows the evolution of DL microstructure reconstruction. As of epoch 160, it has captured some patterns in the microstructure, but the DL reconstruction do not match very well with the 2D smooth microstructures.



(a) Epoch 1 **(b) Epoch 20** **(c) Epoch 40**

(d) Epoch 60 **(e) Epoch 80** **(f) Epoch 100**

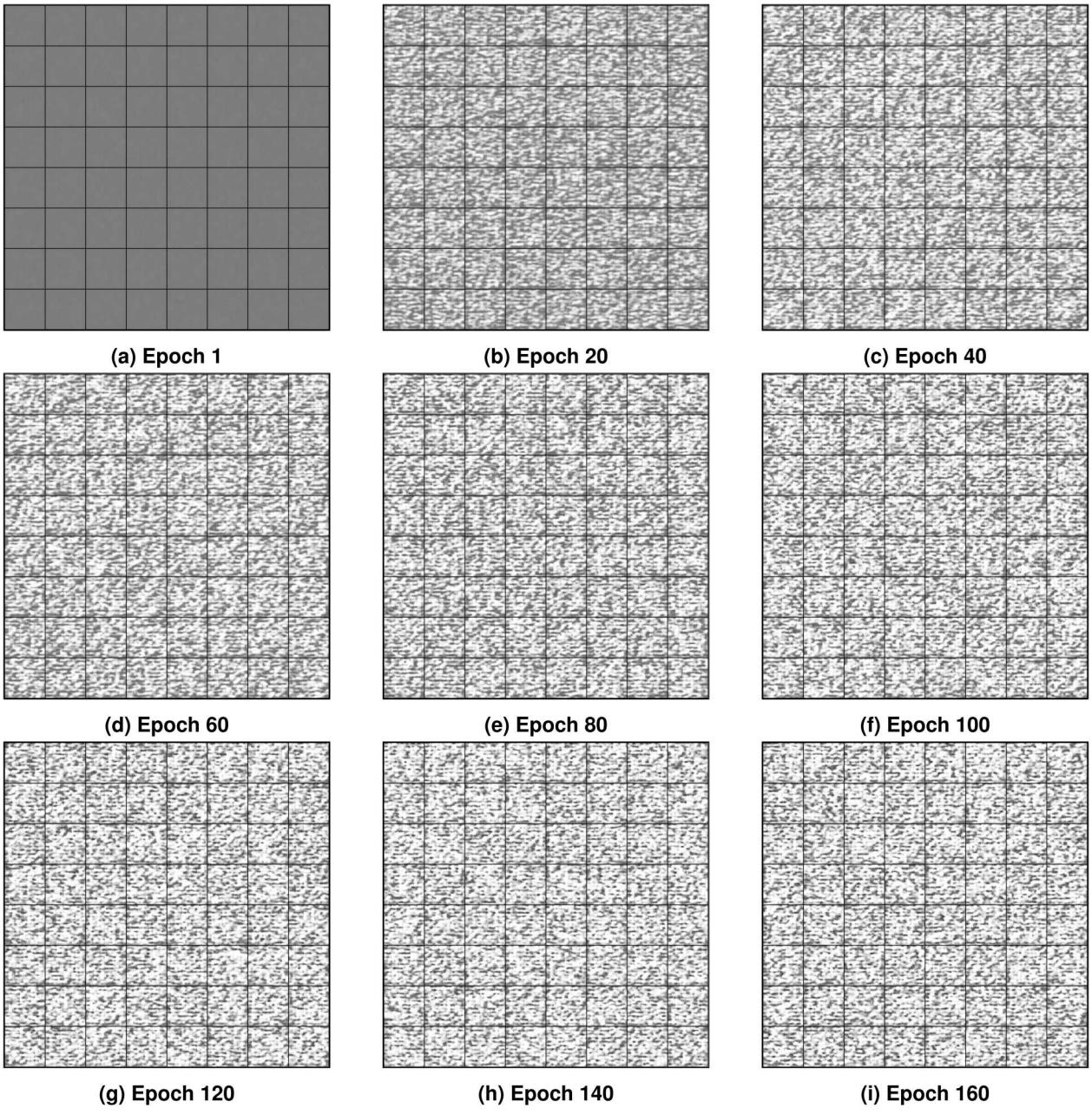
(g) Epoch 120 **(h) Epoch 140** **(i) Epoch 160**

Figure 5-2 8 reconstructions of 2D synthetic smooth microstructures from AE at different epochs.

6. AUTOENCODER FOR SYNTHETIC 2D BINARY MICROSTRUCTURES

The following table describes the DL architecture for AE developed for 2D binary microstructures, which has 15,833 hyper-parameters. Conv2d and ConvTranspose2d layers are used extensively to build the AE.

```
Total number of params: 15833
Total number of trainable params: 15833
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 43, 43]             160
       BatchNorm2d-2          [-1, 16, 43, 43]              32
```



(a) Epoch 1 (b) Epoch 20 (c) Epoch 40

(d) Epoch 60 (e) Epoch 80 (f) Epoch 100

(g) Epoch 120 (h) Epoch 140 (i) Epoch 160

Figure 5-3 64 samples of 2D synthetic smooth microstructures from AE at different epochs.

```
              ReLU-3          [-1, 16, 43, 43]               0
         MaxPool2d-4          [-1, 16, 21, 21]               0
            Conv2d-5          [-1, 32, 11, 11]           4,640
       BatchNorm2d-6          [-1, 32, 11, 11]              64
              ReLU-7          [-1, 32, 11, 11]               0
         MaxPool2d-8          [-1, 32, 10, 10]               0
   ConvTranspose2d-9          [-1, 16, 21, 21]           4,624
             ReLU-10          [-1, 16, 21, 21]               0
  ConvTranspose2d-11           [-1, 8, 65, 65]           6,280
             ReLU-12           [-1, 8, 65, 65]               0
  ConvTranspose2d-13         [-1, 1, 128, 128]              33
            Tanh-14         [-1, 1, 128, 128]               0
================================================================
Total params: 15,833
Trainable params: 15,833
Non-trainable params: 0
```

Figure 6-1 show 8 different binary microstructures using the aforementioned AE, which also demonstrates that the AE has fully captured the information associated with the binary microstructures. The top rows (resulted from SPPARKS simulations) and the bottom rows (from AE) are nearly identical, showing a good match between physics-based and DL reconstruction.

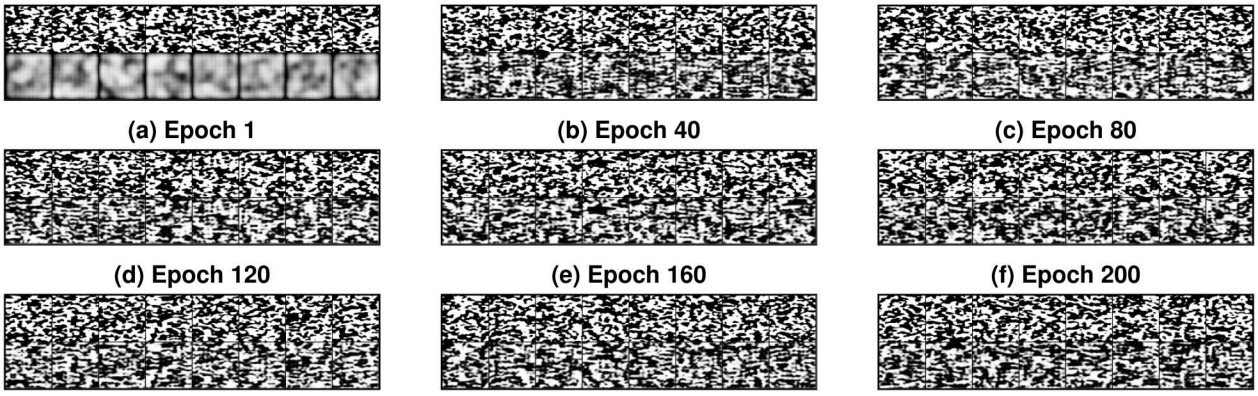
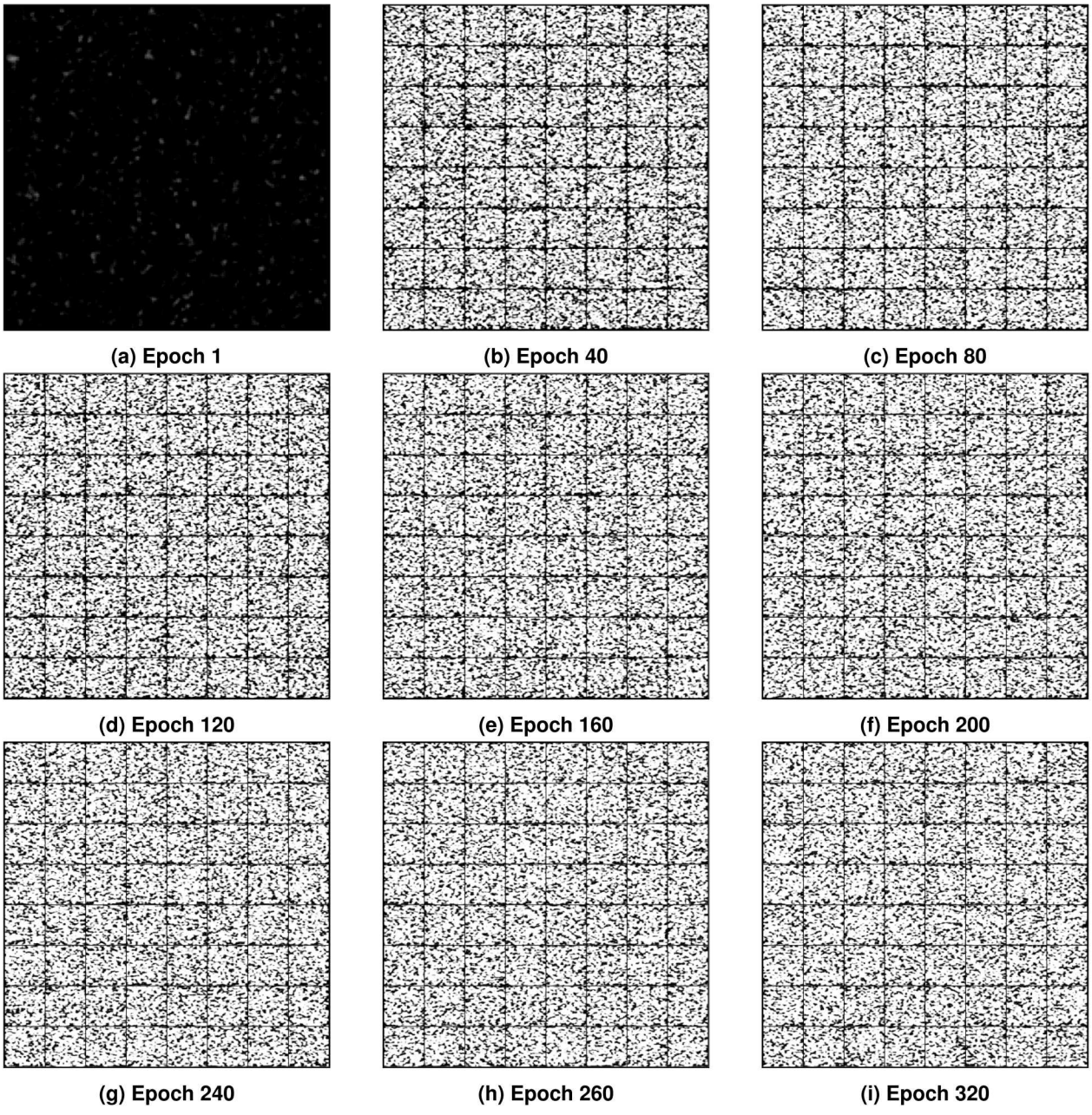


Figure 6-1 8 reconstructions of 2D synthetic binary microstructures from AE at different epochs.

7. AUTOENCODER FOR KMC/GG MICROSTRUCTURES

Typically, AE is composed of an encoder and a decoder. Both of which are CNN. On one hand, the encoder is tasked with "compressing" information from a high-dimensional space to a low-dimensional space which is usually referred to as the latent space, mainly through convolution layers, activation layers, and pooling layers. More advanced layers are also used to enhance its



(a) Epoch 1 (b) Epoch 40 (c) Epoch 80

(d) Epoch 120 (e) Epoch 160 (f) Epoch 200

(g) Epoch 240 (h) Epoch 260 (i) Epoch 320

Figure 6-2 64 samples of 2D synthetic binary microstructures from AE at different epochs.

performance. The decoder, on another hand, is used to "decompress" information from the latent space back to the normal state.

By itself, the trained decoder is a generative deep learning model, in the sense that one can sample from the latent space and generate objects. The objects in this case is the microstructures obtained from SPPARKS. That being said, this approach allows one to generate and reconstruct microstructures if the dataset is sufficient. Fortunately, SPPARKS is well-known to be fairly computationally cheap, thanks to careful parallel implementation on high-performance computing platforms. With the computational power at Sandia, one can reliably generate a substantial dataset within a reasonable time frame.

The architecture of the AE is described in Figure 7-1. We employed a straightforward implementation of AE, where Conv2d and Unconv2d layers are employed extensively.

Figure 7-1 AE DL architecture for kMC/GG microstructure.

The hyper-parameters are described using torchsummary

```
Total number of params: 8793
Total number of trainable params: 8793
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 43, 43]             160
              ReLU-2          [-1, 16, 43, 43]               0
         MaxPool2d-3          [-1, 16, 21, 21]               0
            Conv2d-4           [-1, 8, 11, 11]           1,160
              ReLU-5           [-1, 8, 11, 11]               0
         MaxPool2d-6           [-1, 8, 10, 10]               0
            Conv2d-7             [-1, 4, 5, 5]             292
              ReLU-8             [-1, 4, 5, 5]               0
         MaxPool2d-9             [-1, 4, 4, 4]               0
  ConvTranspose2d-10            [-1, 16, 9, 9]             592
             ReLU-11            [-1, 16, 9, 9]               0
  ConvTranspose2d-12           [-1, 8, 29, 29]           6,280
             ReLU-13           [-1, 8, 29, 29]               0
  ConvTranspose2d-14           [-1, 4, 57, 57]             292
             ReLU-15           [-1, 4, 57, 57]               0
  ConvTranspose2d-16         [-1, 1, 112, 112]              17
             Tanh-17         [-1, 1, 112, 112]               0
================================================================
Total params: 8,793
Trainable params: 8,793
Non-trainable params: 0
```

Figures 7-2 and 7-3 describe the reconstruction of microstructure images and samples at different epochs, respectively. It is clear that the latent variables indeed do exist to describe the microstructure.

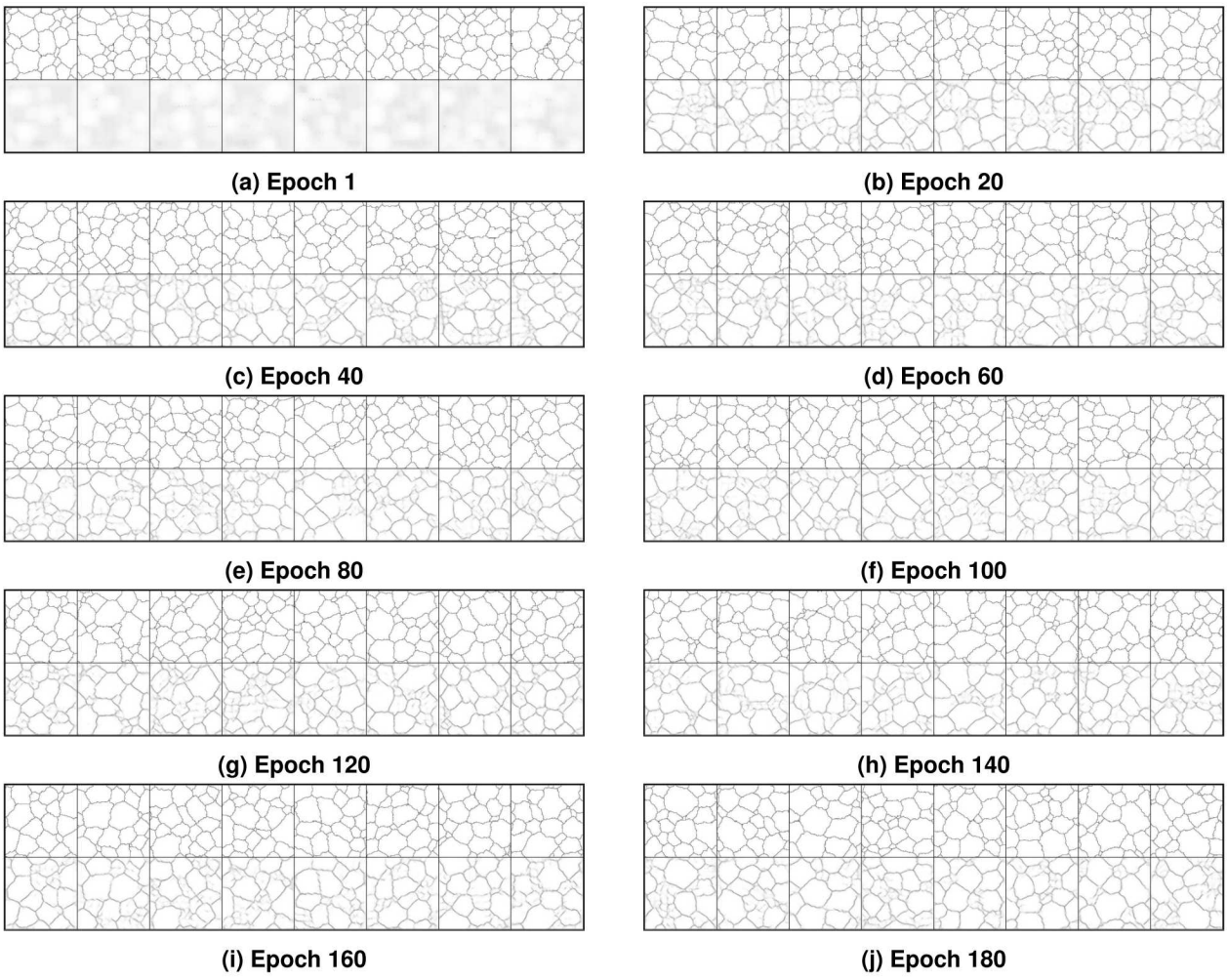
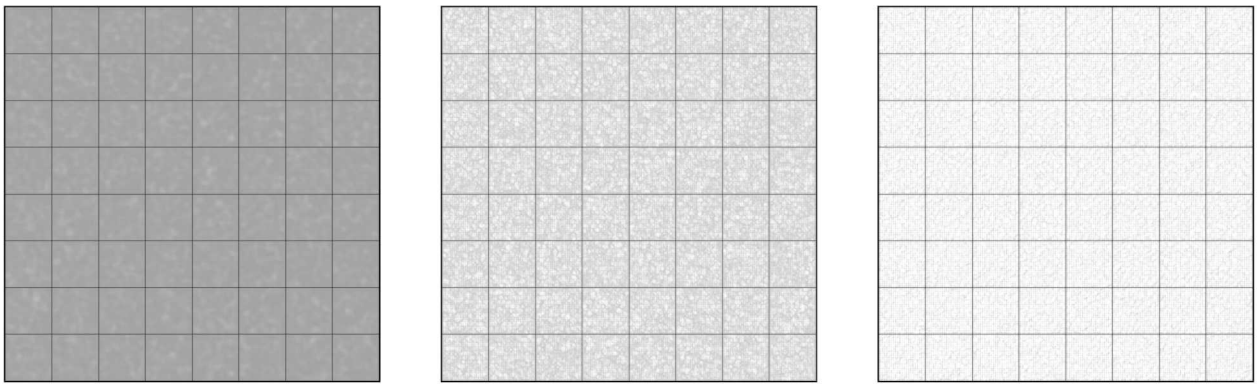


Figure 7-2 8 reconstructions of kMC/GG from AE at different epochs.

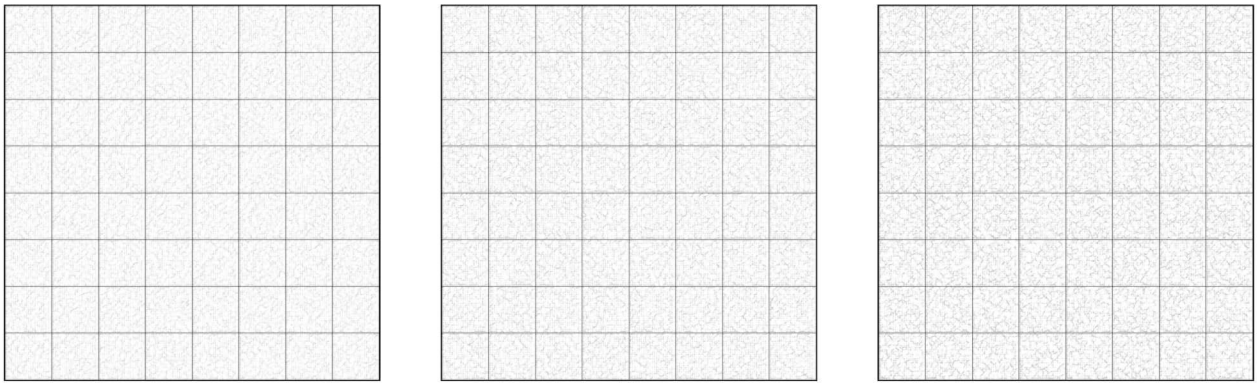
Figure 7-3 shows 64 DL synthetic microstructures, sampled from the latent space, from the learned decoder of the AE. Some reasonable microstructures are observed, but it does not reflect the physical intuition that the grain boundary should be connected together. This challenging question remains unsolved for further future work.

8. AUTOENCODER FOR KMC/ADDITIVE DATASET

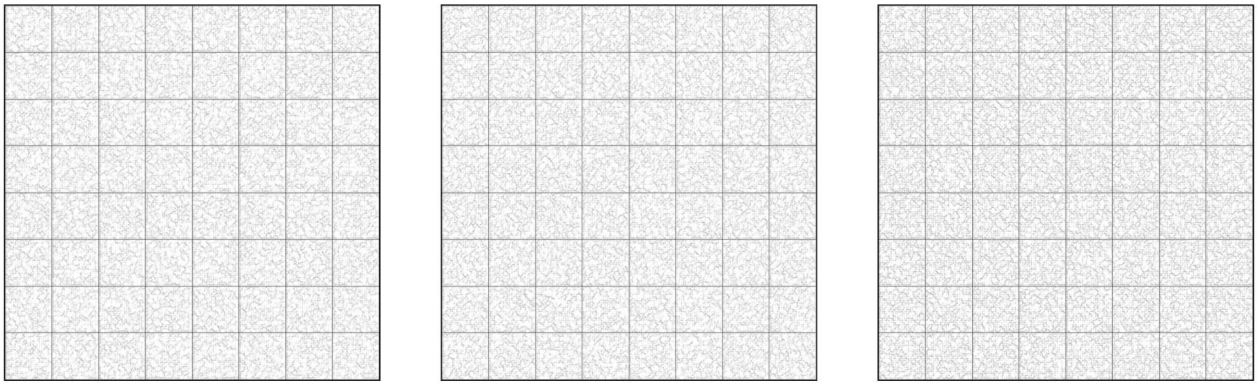
The DL architecture for AE, which is composed of 14,847 hyper-parameters, is shown in the following table. Again, Conv2d and ConvTranspose2d layers are extensively used.



(a) Epoch 1 (b) Epoch 20 (c) Epoch 40



(d) Epoch 60 (e) Epoch 80 (f) Epoch 100



(g) Epoch 120 (h) Epoch 140 (i) Epoch 160

Figure 7-3 64 samples of kMC/GG from AE at different epochs.

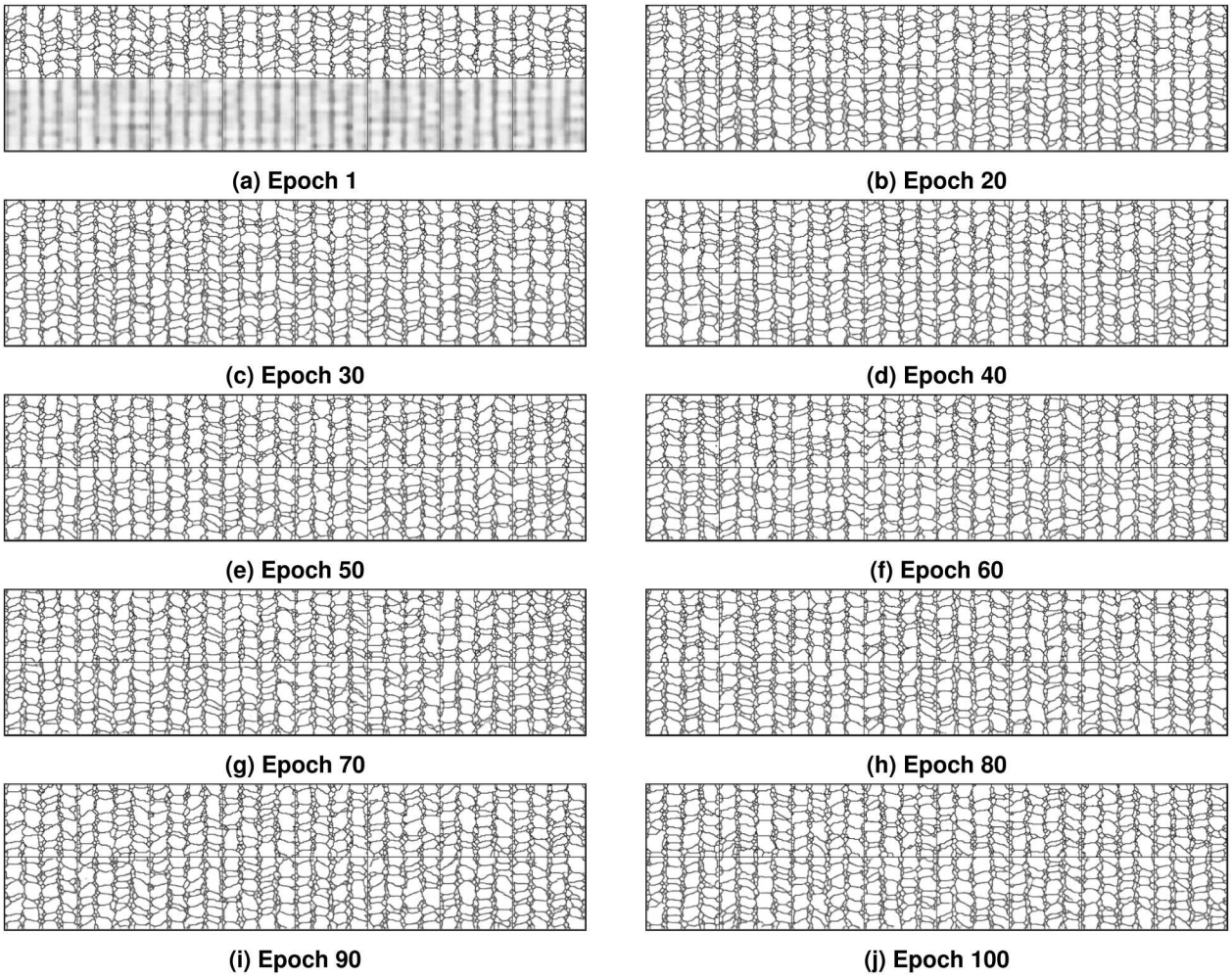
```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 4, 43, 43]              40
              ReLU-2           [-1, 4, 43, 43]               0
         MaxPool2d-3           [-1, 4, 21, 21]               0
            Conv2d-4           [-1, 8, 11, 11]             296
              ReLU-5           [-1, 8, 11, 11]               0
         MaxPool2d-6           [-1, 8, 10, 10]               0
            Conv2d-7           [-1, 16, 5, 5]            1,168
              ReLU-8           [-1, 16, 5, 5]                0
         MaxPool2d-9           [-1, 16, 4, 4]                0
           Conv2d-10           [-1, 32, 3, 3]            2,080
             ReLU-11           [-1, 32, 3, 3]                0
        MaxPool2d-12           [-1, 32, 2, 2]                0
  ConvTranspose2d-13           [-1, 16, 5, 5]            4,624
             ReLU-14           [-1, 16, 5, 5]                0
  ConvTranspose2d-15           [-1, 8, 17, 17]           6,280
             ReLU-16           [-1, 8, 17, 17]               0
  ConvTranspose2d-17           [-1, 4, 33, 33]             292
             ReLU-18           [-1, 4, 33, 33]               0
  ConvTranspose2d-19           [-1, 2, 64, 64]              34
             ReLU-20           [-1, 2, 64, 64]               0
  ConvTranspose2d-21         [-1, 1, 128, 128]              33
             Tanh-22         [-1, 1, 128, 128]               0
================================================================
Total params: 14,847
Trainable params: 14,847
Non-trainable params: 0
```

Figure 8-1 shows 8 different reconstructed microstructures at various epochs. At epoch 100, the AE is able to reconstruct with fine details for kMC/additive dataset, where the top row (DL solutions) is nearly identical with the bottom row (SPPARKS simulated microstructures). This fundamentally proves the concept of extracting low-dimensional non-linear manifold by AE.

Figure 8-2 shows 64 random microstructures, where the latent space is sampled and the trained decoder is used to reconstruct the microstructure. Again, this shows that the AE was able to learn, but not with a fine details as simulations. This challenging problem is posed as future work.

9. AUTOENCODER FOR UHCSDB DATASET 48X64

9.1. Data curation



(a) Epoch 1

(b) Epoch 20

(c) Epoch 30

(d) Epoch 40

(e) Epoch 50

(f) Epoch 60

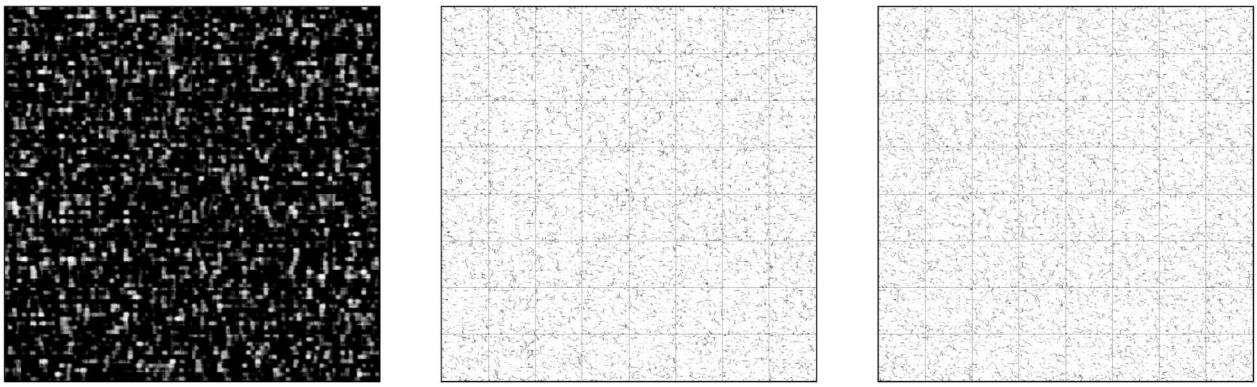
(g) Epoch 70

(h) Epoch 80

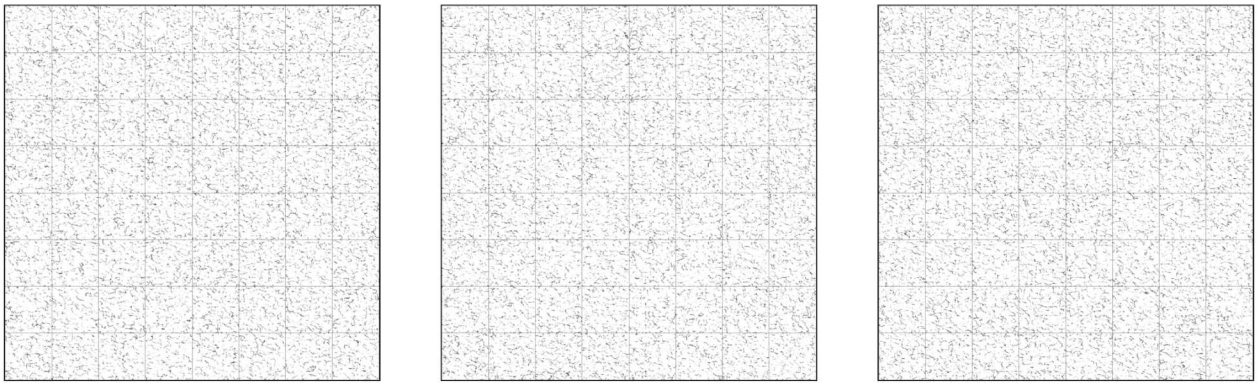
(i) Epoch 90

(j) Epoch 100

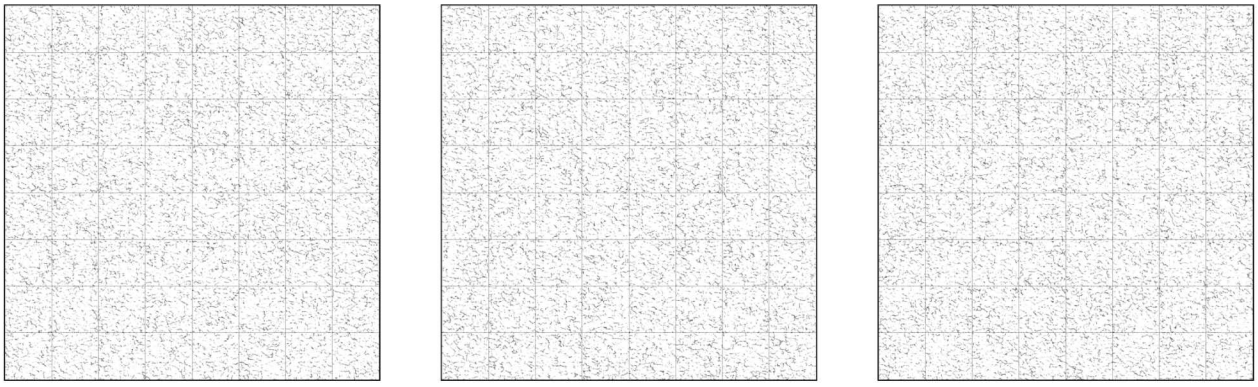
Figure 8-1 8 reconstructions of kMC/additive from AE at different epochs.



(a) Epoch 1 (b) Epoch 20 (c) Epoch 40



(d) Epoch 60 (e) Epoch 80 (f) Epoch 100



(g) Epoch 120 (h) Epoch 140 (i) Epoch 160

Figure 8-2 64 samples of kMC/additive from AE at different epochs.

```
## extractMetadata.sh
#!/bin/bash

for folderName in $(ls -1dv */); do
  cd $folderName
  rm -fv minorMetadata.csv minor-metadata.csv

  for cropImgName in $(ls -1v *.png); do
    micrographName=$(echo $cropImgName | cut -d. -f2)
    # grep "${micrographName}\." ../uhcs-metadata.csv >> minor-metadata.csv
    micrographid=$(echo $micrographName | tr -dc '0-9')
    lineNum=$( echo "${micrographid} + 1" | bc -l)
    sed -n ${lineNum}p ../uhcs-metadata.csv >> minor-metadata.csv
  done

  cd ..
  echo "done $folderName"
done
```

But before performing these steps, we must rescale the micrograph to be on a same scale, because otherwise, one pixels in one micrograph is not as the same as another. We approach this problem by firstly cropping the annotation of the micrograph, which include the magnification, scale bar, and other information relevant to the micrograph. Note that the metadata is also supported by another separate file uhcs-metadata.csv, which contains identical information regarding the dataset. After being cropped, all micrographs have the same dimension of 645 pixels $\times$ 484 pixels.

```
import numpy as np
import matplotlib.pyplot as plt
# 'uhcs-metadata.csv' or 'minor-metadata.csv'
metadata = np.loadtxt('minor-metadata.csv', delimiter=',', dtype=str)
magnArr = metadata[:,9]

# https://stackoverflow.com/questions/4289331/how-to-extract-numbers-from-a-string-in-python

magnArrInt = []
for j in range(len(magnArr)):
  magn = ''.join([i for i in magnArr[j] if i.isdigit()])
  if magn != '':
    magn = int(magn)
    if isinstance(magn, int):
      magnArrInt += [magn]

magnArrInt = np.array(magnArrInt)
print('max magn = %d' % np.max(magnArrInt))
print('min magn = %d' % np.min(magnArrInt))
print('median magn = %d' % np.median(magnArrInt))

plt.hist(magnArrInt, bins='auto')
plt.xlabel('magnification', fontsize=24)
plt.ylabel('frequency', fontsize=24)
plt.title('magnification analysis', fontsize=30)
plt.show()
```

Second, using the magnification, we convert the image to roughly the same scale. We remove all the data without labels, as there is no processing information associated with the microstructure. This significantly reduces the number of usable micrographs to 1024. The wide range of magnification from 35X to 63833X makes a selection for a uniform magnification challenging. To demonstrate the idea, we choose the dataset spheroidite.* to work with, since it contains the most micrographs.

```
#!/bin/bash
for type in martensite network pearlite+spheroidite pearlite+widmanstatten pearlite spheroidite+widmanstatten spheroidite; do
  mkdir -p ${type}-unified/
done

for fN in $(ls -1dv *.*/); do
  type=$(echo $fN | cut -d. -f1)
  cp -rfv $fN/*.png ${type}-unified
done
```

The magnification range of this sub-dataset is from 246X to 19641X. The histogram of magnification is shown in Figure 9-1. Based on the analysis in Figure 9-1, the median of 1964X is used as the uniform scale for this sub-dataset spheroidite. The min and max magnification is 246 and

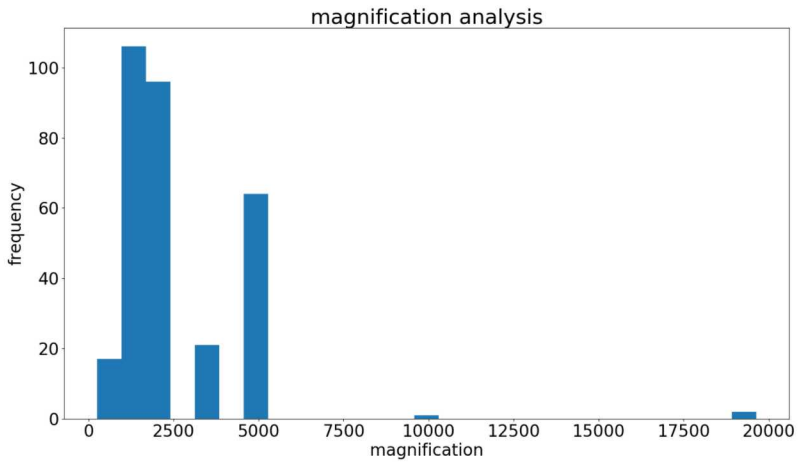


Figure 9-1 Magnification analysis for spheroidite.

1964, respectively. Also, we remove micrographs where the magnification is not available, e.g. micrograph762.

```
import glob, os
import numpy as np
from PIL import Image
# https://stackoverflow.com/questions/273946/how-do-i-resize-an-image-using-pil-and-maintain-its-aspect-ratio

metadata = np.loadtxt('minor-metadata.csv', dtype=str, delimiter=',')
imList = glob.glob('*.png')
targetMagn = 1964 # median

# imName = 'cropped.micrograph1295.png'
for imName in imList:
  print('imName = %s' % imName) # debug
  micrographName = imName.split('.')[1] # micrograph1637
  im = Image.open(imName)
  width, height = im.size # 645, 484
  aspectRatio = height/width
  print('micrographName = %s' % micrographName)
  rowIndex = np.where(metadata[:, 14] == micrographName + '.tif') # search for the row
  # print(rowIndex)
  # print(metadata[rowIndex, 14])
  # print(rowIndex)
  # print(len(rowIndex[0]))

  if len(rowIndex[0]) == 0:
    rowIndex = np.where(metadata[:, 14] == micrographName + '.png') # search for another extension instead
    print('Searching for %s' % (micrographName + '.png'))
  if len(rowIndex[0]) == 0:
    rowIndex = np.where(metadata[:, 14] == micrographName + '.bmp') # search for another extension instead
    print('Searching for %s' % (micrographName + '.bmp'))

  rowIndex = int(rowIndex[0])
  print('rowIndex = %d; imName = %s' % (rowIndex, imName))
  magn = ''.join([i for i in metadata[rowIndex, 9] if i.isdigit()])
  if magn != '':
    # print(magn)
    magn = int(magn)
    # principles: (1) enlarge img with small magn, (2) reduce img with large magn
    scaleRatio = targetMagn / magn
    targetWidth = int(width * scaleRatio)
    targetHeight = int(height * scaleRatio)
    # scale image
    rescaledIm = im.resize((targetWidth, targetHeight), Image.ANTIALIAS)
    im.save('rescaled.' + micrographName + '.png')
    print('done %s\n' % imName)
```

This step creates a folder named rescaled with 344 microstructure images for spheroidite with various sizes correspond to various image quality. We also remove rescaled.micrograph{1207,1173}.png due to their low quality and small size.

9.2. Random crop

From the 344 microstructure images with various sizes, we perform random crop, where the number of random crop is proportional to images' area. Recall that the standard image is of 1964X, where width and height are 645 and 484 pixels, respectively. The smallest size of rescaled images is (48, 64), whereas the largest size of the rescaled images is (1936, 2540). The median of rescaled images is (484, 645). We take 48×64 as a unit patch for random crop operation. The number of crop is calculated as

$$\#\text{ of crops} = \left\lfloor \frac{\text{imageHeight} \times \text{imageWidth}}{\text{patchHeight} \times \text{patchWidth}} \times r \right\rfloor, \tag{2}$$

where r is a number of redundancy which should be chosen depending on the size of generated dataset. If the dataset is too small, r should be increased, and vice versa. This allows us to subsample 344 micrographs to a dataset with 56392 images, which is acceptable for DL.

```
## randomCrop.py

import numpy as np
import os, glob
import skimage.io as io

defaultHeight = 48
defaultWidth = 64

# run in .../dataset/uhcsdb/nistRepo/randomCropped-SameScale-croppedMicrographs/spheroidite-unified
# make sure 'rescaled/' is available (as input folder)
# and 'randomCrop/' is available (as output folder)

numOfPatch = 0
for imgName in glob.glob('./rescaled/*.png'):
  img = io.imread(imgName)
  width, height = img.shape
  numOfCrops = np.floor( width * height / defaultWidth / defaultHeight * 1.5)
  print('randomCrop %s: h = %d, w = %d: numOfCrops = %d' % (imgName, height, width, numOfCrops))
  numOfPatch += numOfCrops

print('Total number of patches: %d' % (numOfPatch))
```

We then randomly crop the images to more than 100,000 patches of 64×48 and subsequently remove low contrast patches, as well as faulty patches, e.g. patches that after being saved does not retain its size anymore.

```
## randomCrop.py
import numpy as np
import os, glob
# import skimage.io as io
import skimage

from PIL import Image

defaultHeight = 48
defaultWidth = 64

# run in .../dataset/uhcsdb/nistRepo/randomCropped-SameScale-croppedMicrographs/spheroidite-unified
# make sure 'rescaled/' is available (as input folder)
# and 'randomCrop/' is available (as output folder)

logFile = open('randomCrop.log', 'w')
numOfPatch = 0
patchID = 0
os.system('mkdir -p randomCrop/')
for imgName in glob.glob('./rescaled/*.png'):

  # read images
  img = skimage.io.imread(imgName) # sklearn
  # img = Image.open(imgName) #  PIL

  imgName = imgName.split('/')[-1]
  imgName = imgName.split('.')[1]
  width, height = img.shape # sklearn
  # width, height = img.size # PIL
```

```
  # determine how many patches to save
  redundantRatio = 1.5
  numOfCrops = np.floor( width * height / defaultWidth / defaultHeight * redundantRatio)
  # numOfCrops = 1 # debug

  numOfCrops = int(numOfCrops)
  print('randomCrop %s: h = %d, w = %d: numOfCrops = %d' % (imgName, height, width, numOfCrops))
  numOfPatch += numOfCrops

  saveError = True
  isNotLowContrast = True
  while saveError and isNotLowContrast: # no save error AND not low contrast
    for i in range(numOfCrops):
      if height > defaultHeight and width > defaultWidth:
        sH = np.random.randint(0, height - defaultHeight) # starting height
        sW = np.random.randint(0, width - defaultWidth) # starting width
      else:
        sH = 0
        sW = 0

      # crop images
      cropImg = img[sH:sH+defaultHeight, sW:sW+defaultWidth] # sklearn
      # cropImg = img.crop((sH, sW, sH + defaultHeight, sW + defaultWidth)) # PIL

      cropImgName = 'patch%d' % patchID

      # check low contrast
      # # isNotLowContrast = not skimage.exposure.is_low_contrast(cropImg)
      # print('isNotLowContrast = %r' % isNotLowContrast)

      # save images
      try:
        skimage.io.imsave('./randomCrop/%s.png' % cropImgName, cropImg, check_contrast=False) # sklearn
        # cropImg.save('./randomCrop/%s.png' % cropImgName) # PIL
        saveError = False
        patchID += 1
      except:
        saveError = True


    logFile.write('randomCrop %s patch %s in i=%d/numOfCrops=%d: h = %d, w=%d at sH = %d, sW = %d\n' % (imgName, patchID, i, numOfCrops, height, width,


logFile.close()
print('Total number of patches: %d' % (numOfPatch))
```

```
## removeFaultPatch.py

import numpy as np
import os, glob
# import skimage.io as io
import skimage

from PIL import Image

defaultHeight = 48
defaultWidth = 64

targetFolder = 'spheroidite-unified-randomCrop'
imgList = glob.glob('%s/*.png' % targetFolder)
for imName in imgList:
  # im = Image.open(imName)
  im = skimage.io.imread(imName)
  # w, h = im.size
  h, w = im.shape
  if w != defaultWidth or h != defaultHeight:
    print('%s: h=%d, w=%d' % (imName, h, w))
    os.system('rm -v %s' % imName)
  else:
    print('Pass %s: h=%d, w=%d.' % (imName, h, w))
```

9.3. Dataset

The dataset is split into 74581 patches for training and 21627 patches for testing. The dimension of the patch is 64 pixels$\times$48 pixels.

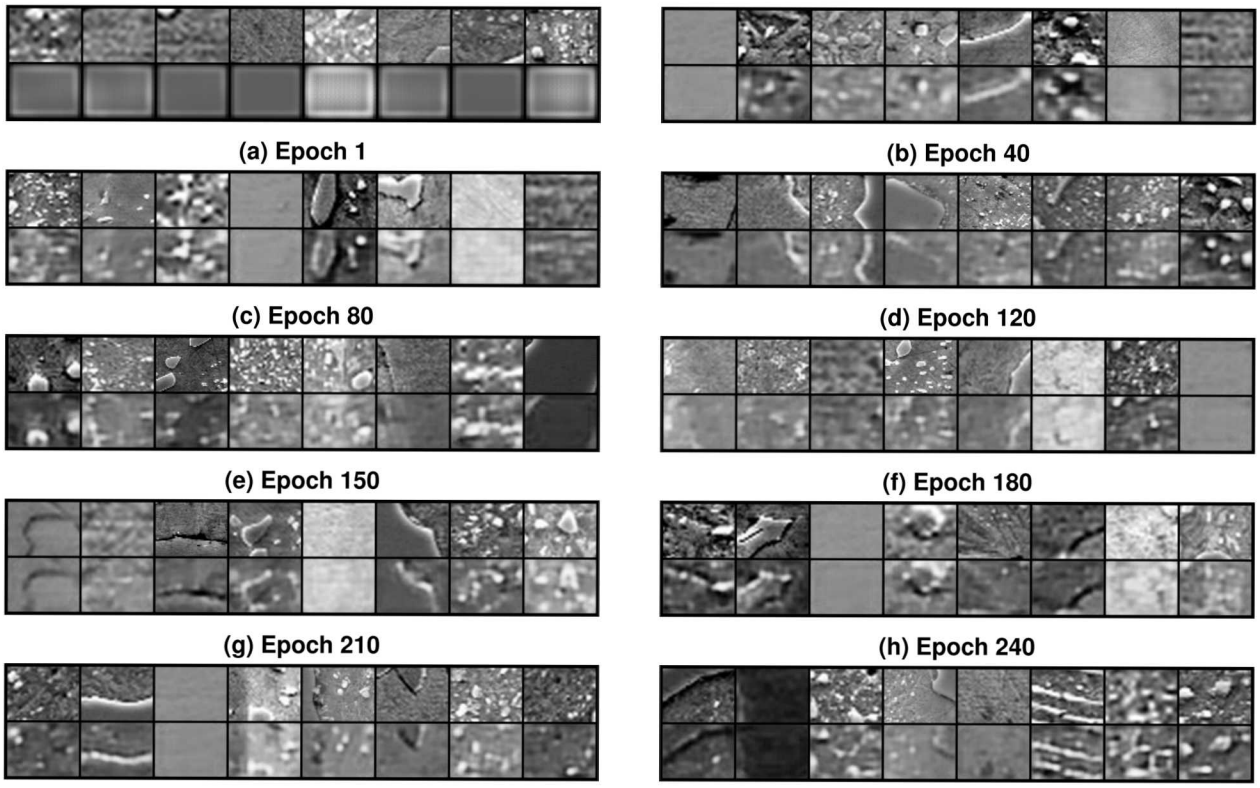
9.4. UHCSDB autoencoder 48x64

```
Total number of params: 5855
Total number of trainable params: 5855
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 2, 48, 64]              20
              ReLU-2           [-1, 2, 48, 64]               0
         MaxPool2d-3           [-1, 2, 46, 62]               0
            Conv2d-4           [-1, 4, 46, 62]              76
              ReLU-5           [-1, 4, 46, 62]               0
         MaxPool2d-6           [-1, 4, 44, 60]               0
            Conv2d-7           [-1, 8, 22, 30]             296
              ReLU-8           [-1, 8, 22, 30]               0
         MaxPool2d-9           [-1, 8, 20, 28]               0
           Conv2d-10           [-1, 8, 10, 14]             584
             ReLU-11           [-1, 8, 10, 14]               0
        MaxPool2d-12            [-1, 8, 8, 12]               0
  ConvTranspose2d-13            [-1, 8, 8, 12]             584
             ReLU-14            [-1, 8, 8, 12]               0
  ConvTranspose2d-15           [-1, 8, 10, 14]           1,608
             ReLU-16           [-1, 8, 10, 14]               0
  ConvTranspose2d-17           [-1, 8, 21, 29]           1,608
             ReLU-18           [-1, 8, 21, 29]               0
  ConvTranspose2d-19           [-1, 4, 43, 59]             804
             ReLU-20           [-1, 4, 43, 59]               0
  ConvTranspose2d-21           [-1, 2, 45, 61]             202
             ReLU-22           [-1, 2, 45, 61]               0
  ConvTranspose2d-23           [-1, 1, 48, 64]              73
             Tanh-24           [-1, 1, 48, 64]               0
================================================================
Total params: 5,855
Trainable params: 5,855
Non-trainable params: 0
```

9.5. Reconstruction

Figure 9-2 shows 8 reconstructions of random microstructure images at different epochs. It is observed that the AE proposed can capture some, but not to a great detail of the microstructure. Perhaps a deeper architecture is suitable for this purpose. Figure 9-3 shows 2 samples at different epochs using the same AE architecture. Even though the quality of the images in Figure 9-2 is reasonably good and its training converges quickly, the decoder is incapable of generating reasonable

microstructures. This limitation of AE prompts us to another DL architecture, called GAN, which is also explored in the section below.



(a) Epoch 1 (b) Epoch 40

(c) Epoch 80 (d) Epoch 120

(e) Epoch 150 (f) Epoch 180

(g) Epoch 210 (h) Epoch 240

(i) Epoch 270 (j) Epoch 300

Figure 9-2 8 reconstructions of UHCSDB from AE at different epochs.

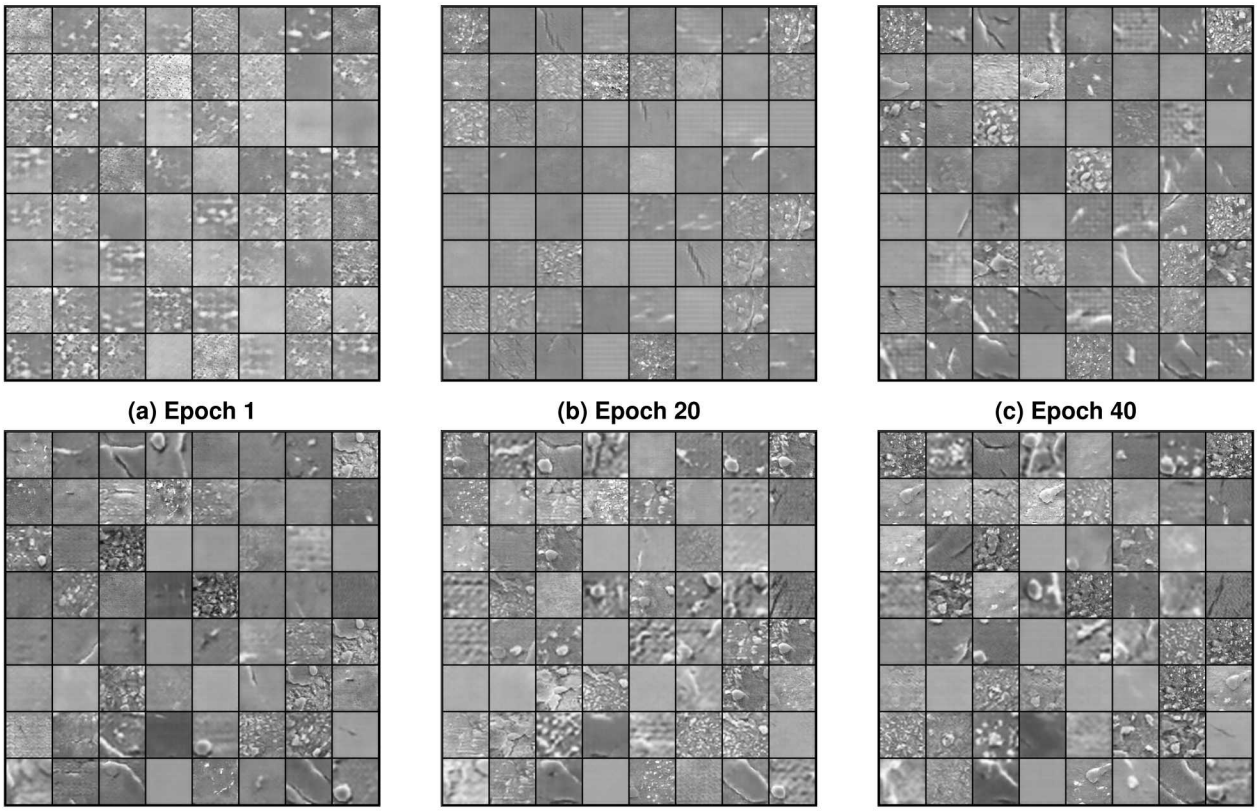
(a) Epoch 270 (b) Epoch 300

Figure 9-3 2 samples of UHCSDB from AE at different epochs.

10. GAN FOR UHCSDB DATASET 64X64

We subsample the previous UHCSDB dataset, this time with the patch size of 64×64. 90000 patches are used as the training dataset, whereas 38940 patches are used as the testing dataset.

DCGAN [14] is employed in this work. Figure 10-1 shows the microstructure generated at different epochs.



(a) Epoch 1 (b) Epoch 20 (c) Epoch 40

(d) Epoch 60 (e) Epoch 80 (f) Epoch 100

Figure 10-1 6 collection of 64 samples of UHCSDB from GAN at different epochs.

The quality of generated microstructures has gradually improved over the training; by the epoch 100, it was capable of generating microstructures that looks aesthetically similar to the experimental microstructure. However, DCGAN also suffers from mode collapse, as in the sense that it generates many similar images. Training GAN is well-known to be a challenging problem, which will require further work.

11. DISCUSSION

In this project, DL is implemented via PyTorch. Sandia Blake testbed is utilized for most of the training. We perform experiments with various activation functions, loss functions, and architectures, including AE and GAN for reconstructing microstructure.

The AE was successfully implemented for the synthetic 2D microstructure dataset. We found that the MaxPool layer performs better than the AvgPool layer for this dataset, given the same DL architecture and loss function. VAE has been shown to be very challenging to train, while AE offers a fairly straightforward to implement. Perhaps one of the reasons is because the difference in terms of loss functions: AE employs mean-square error (MSE).

We also observe that the utilization of BatchNorm layer also increases the training speed and improves the efficiency of optimization in seeking optimal hyper-parameters. The traditional approach is Conv/ReLU/MaxPool can be transformed to Conv/BatchNorm/ReLU/MaxPool. There has been some debates regarding the location of the BatchNorm layer, and the majority advocates its location right after the convolution layer.

An important lesson learned from brute-force of DL application on CPFEM dataset is that the input domain is simply too large to apply DL methods blindly. Yet, we fail to see it upfront. Theoretically, there is no difference, but practically, the difference is huge (as the problem is nearly computationally intractable). Dissecting the large dataset into smaller pieces based on localization assumption may work better.

The reconstruction capability of VAE and GAN has been much a debate, even though in practice, GAN has the advantage of generating more realistic samples. VAE is widely supported theoretically, as there is a clear way to evaluate the quality of the model by the log-likelihood, either estimated by importance sampling or lower-bounded. VAEs tend to generate more blurred samples compared to those from GANs, while GANs could suffer from mode collapse in multi-modal density.

We also found that the quality of the reconstruction is (highly) dependent on the number of channels in the very first convolutional layer. 16 channels is sufficient for kMC/GG dataset.

12. CONCLUSION

As DL is emerging, DL applications in materials science and other fields are generally a new topic for research and development. DL has not matured yet. Recent years has observed a progressive development of GAN with higher and higher resolution, so adapting to the new technology is certainly a challenge. Much work remains to be done in materials science, both computationally and experimentally.

In this report, we describe our recent and preliminary efforts to develop two tracks of DL in materials science, including both supervised and unsupervised DL. For supervised learning, we proposed CPNet, which is a 3D-CNN DL architecture for solving CPFEM. Reasonable agreement is observed, which paves ways for future research. For unsupervised learning, we generated 5-6 synthetic microstructure datasets from SPPARKS/kMC, and adopted UHCSDB dataset, which is experimental.

Promising results are achieved. In supervised learning, a DL algorithm is achieved to predict materials response with a limited dataset. In unsupervised learning, good microstructure generation and reconstruction are achieved in some cases. Both topics are not mature; much work remains to be done, but at least this report has proved beyond reasonable doubt that DL could have a lasting impact to the perception of materials science community in general.

ACKNOWLEDGMENT

Anh Tran thank Joseph Bishop for mentoring this project and for his helpful and inspiring discussions, and Prof. Surya Kalidindi (Georgia Tech) and Dr. Yuksel Yabansu (Georgia Tech) for sharing their CPFEM dataset. The support from the LDRD Exploratory Express, managed by Dr. Hy Tran, is gratefully acknowledged.

APPENDIX A BLAKE ADVANCED TESTBED (CPU/SLURM SCHEDULER)

In this section, we describe the hardware and software installation on Blake testbed.

```
$ lscpu

Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                96
On-line CPU(s) list:   0-95
Thread(s) per core:    2
Core(s) per socket:    24
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz
Stepping:              4
CPU MHz:               2101.000
BogoMIPS:              4194.85
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              33792K
NUMA node0 CPU(s):     0-23,48-71
NUMA node1 CPU(s):     24-47,72-95

$ module list
Currently Loaded Modulefiles:
  1) cmake/3.12.3              5) numa/2.0.11              9) readline/8.0.0          13) ncurses/6.0.0
  2) zlib/1.2.11               6) papi/5.5.1              10) python/3.7.3
  3) binutils/2.30.0           7) java/oracle/1.8.0       11) bzip2/1.0.6
  4) gcc/7.2.0                 8) openmpi/2.1.2/gcc/7.2.0 12) git/2.9.4
```

Note that module python/3.7.3 must be used in order to run PyTorch. TPLs to be installed:

- torchsummary
- torchviz
- hiddenlayer

Visualizing the model (note that torch may have to be upgraded)

- via torchviz (link to GitHub)

```
from torchviz import make_dot
x = torch.randn(1, 1, imgDim, imgDim)
y = model(x)
make_dot(y, params=dict(list(model.named_parameters()))).render()
```

- via hiddenlayer (link to GitHub)

```
x = torch.zeros([1, 1, 128, 128])
import hiddenlayer as hl
g = hl.build_graph(model, x)
g = g.build_dot()
g.render('test', view=True, format='png')
```

APPENDIX B WHITE ADVANCED TESTBED (GPU/LSF SCHEDULER)

```
The White hardware environment is comprised of three types of compute
node:

(1) POWER8 Tuleta Processors (5-core dual-NUMA-per socket, dual-socket)
    with a single NVIDIA Kepler-K40 GPU per socket connect via PCIe. These
    nodes are provided under the rhel7T queue.

(2) POWER8 Firestone Processors (8-core per socket, dual-socket) with
    a single NVIDIA Kepler-K80 GPU per socket connected via PCIe. These nodes
    are provided under the rhel7F queue.

(3) POWER8+ Firestone Processors (8-core per socket, dual-socket) with
    two NVIDIA Pascal P100 GPUs per socket connected via NVLINK-1. These
    nodes are provided under the rhel7G queue (because they are called
    Garrison/Minsky blades by IBM).

The nodes are interconnected with Mellanox InfiniBand.
```

An official documentation on how to use LSF scheduler is referred here. How to set up Anaconda environment can be read here.

```
1)  cmake/3.12.3
2)  zlib/1.2.8
3)  binutils/2.30.0
4)  gcc/7.2.0
5)  cuda/9.2.88
6)  papi/5.5.1
7)  java/ibm/sdk/8.0.0
8)  openmpi/2.1.2/gcc/7.2.0/cuda/9.2.8
9)  readline/7.0.0
10) python/3.7.3
11) anacoonda3/4.8.2-python-3.7.6
```

Regarding conda and pytorch:

```
# cheat sheet
# https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf
conda create -n whiteConda python=3.6 anaconda
conda activate whiteConda
conda install conda
conda update -n base -c defaults conda
# conda config --append channels conda-forge
conda install numpy scipy matplotlib scikit-learn scikit-image keras ipython django pandas jupyter parameterized Theano mako mpi4py n

# conda install tf tensorflow  tf-gpu tensorflow-gpu
# conda install pytorch torchvision cudatoolkit=10.2 -c pytorch
which pip
# NOTE: White testbed is power8, i.e. ppc64le, not supported officially from pytorch
conda install -c engility pytorch
conda install -c engility torchvision

# conda deactivate
# conda env remove --name whiteConda

conda init bash # conda init --help
conda install -n whiteConda pip
conda info
python3 -s # do not add user site-packages directory to sys.path

# To see the Python path
```

```
python3
import sys
print(sys.path)
```

Regarding LSF scheduler:

```
# https://hpc.llnl.gov/banks-jobs/running-jobs/lsf-commands
bsub
bqueues
bstatus
```

APPENDIX C A PRIMER ON CNN

Here, we implement our DL approach on PyTorch, where the documentations of neural layers are here.

C.1 Conv(1d,2d,3d)

```
(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')
```

$$D_{\text{out}} = \left\lfloor \frac{D_{\text{in}} + 2 \times \text{padding} - \text{dilation} \times (\text{kernelSize} - 1) - 1}{\text{stride}} + 1 \right\rfloor \tag{3}$$

C.2 {AvgPool,MaxPool}(1d,2d,3d)

```
(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)
```

$$D_{\text{out}} = \left\lfloor \frac{D_{\text{in}} + 2 \times \text{padding} - \text{kernelSize}}{\text{stride}} + 1 \right\rfloor \tag{4}$$

C.3 ConvTranspose(1d,2d,3d)

```
(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')
```

$$D_{\text{out}} = (D_{\text{in}} - 1) \times \text{stride} - 2 \times \text{padding} + \text{dilation} \times (\text{kernelSize} - 1) + \text{output\_padding} + 1 \tag{5}$$

C.4 Training

Training methods, including Adadelta, Adagrad, Adam, AdamW, SparseAdam, Adamax, ASGD, LBFGS, RMSprop, Rprop, and SGD, are documented here for PyTorch.

C.5 Standard benchmark dataset

About 25 standard benchmarking datasets are available here. Source code is available on GitHub.

C.6 Parallelization

Parallelization over CPU here

Parallelization over GPU here, here, here, here, tutorial here

APPENDIX D A PRIMER ON VARIATIONAL AUTOENCODERS

Original paper by Diederik and Welling [54]. Tutorials by Doersch [55] and Charte et al [56].

REFERENCES

[1] P. Fernandez-Zelaia, Y. C. Yabansu, S. R. Kalidindi, A comparative study of the efficacy of local/global and parametric/nonparametric machine learning methods for establishing structure–property linkages in high-contrast 3D elastic composites, Integrating Materials and Manufacturing Innovation 8 (2) (2019) 67–81.

[2] M. Campbell, A. J. Hoane Jr, F.-h. Hsu, Deep blue, Artificial intelligence 134 (1-2) (2002) 57–83.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of Go with deep neural networks and tree search, nature 529 (7587) (2016) 484–489.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., Mastering the game of Go without human knowledge, nature 550 (7676) (2017) 354–359.

[5] C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, J. Kepner, A. McCabe, P. Michaleas, J. Mullen, D. O'Gwynn, et al., Driving big data with big compute, in: 2012 IEEE Conference on High Performance Extreme Computing, IEEE, 2012, pp. 1–6.

[6] A. Tran, H. Tran, Data-driven high-fidelity 2D microstructure reconstruction via non-local patch-based image inpainting, Acta Materialia 178 (2019) 207–218.

[7] S. R. Kalidindi, A. Khosravani, B. Yucel, A. Shanker, A. L. Blekh, Data Infrastructure Elements in Support of Accelerated Materials Innovation: ELA, PyMKS, and MATIN, Integrating Materials and Manufacturing Innovation 8 (4) (2019) 441–454.

[8] G. Landi, S. R. Niezgoda, S. R. Kalidindi, Multi-scale modeling of elastic response of three-dimensional voxel-based microstructure datasets using novel DFT-based knowledge systems, Acta Materialia 58 (7) (2010) 2716–2725.

[9] T. Fast, S. R. Kalidindi, Formulation and calibration of higher-order elastic localization relationships using the MKS approach, Acta Materialia 59 (11) (2011) 4595–4605.

[10] A. Iyer, B. Dey, A. Dasgupta, W. Chen, A. Chakraborty, A conditional generative model for predicting material microstructures from processing methods, arXiv preprint arXiv:1910.02133.

[11] S. Chun, S. Roy, Y. T. Nguyen, J. B. Choi, H. Udaykumar, S. S. Baek, Deep learning for synthetic microstructure generation in a materials-by-design framework for heterogeneous energetic materials, arXiv preprint arXiv:2004.04814.

[12] D. Fokina, E. Muravleva, G. Ovchinnikov, I. Oseledets, Microstructure synthesis using style-based generative adversarial networks, Physical Review E 101 (4) (2020) 043308.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.

[14] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv preprint arXiv:1511.06434.

[15] M.-Y. Liu, O. Tuzel, Coupled generative adversarial networks, in: Advances in neural information processing systems, 2016, pp. 469–477.

[16] T. Karras, T. Aila, S. Laine, J. Lehtinen, Progressive growing of gans for improved quality, stability, and variation, arXiv preprint arXiv:1710.10196.

[17] T. Karras, S. Laine, T. Aila, A style-based generator architecture for generative adversarial networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4401–4410.

[18] R. Bostanabad, Reconstruction of 3D microstructures from 2D images via transfer learning, Computer-Aided Design 128 (2020) 102906.

[19] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[20] R. Singh, V. Shah, B. Pokuri, S. Sarkar, B. Ganapathysubramanian, C. Hegde, Physics-aware deep generative models for creating synthetic microstructures, arXiv preprint arXiv:1811.09669.

[21] B. L. DeCost, T. Francis, E. A. Holm, Exploring the microstructure manifold: Image texture representations applied to ultrahigh carbon steel microstructures, Acta Materialia 133 (2017) 30–40.

[22] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, Journal of machine learning research 9 (Nov) (2008) 2579–2605.

[23] L. van Der Maaten, Accelerating t-SNE using tree-based algorithms, The Journal of Machine Learning Research 15 (1) (2014) 3221–3245.

[24] B. L. DeCost, B. Lei, T. Francis, E. A. Holm, High throughput quantitative metallography for complex microstructures using deep learning: a case study in ultrahigh carbon steel, arXiv preprint arXiv:1805.08693.

[25] J. Ling, M. Hutchinson, E. Antono, B. DeCost, E. A. Holm, B. Meredig, Building data-driven models with microstructural images: Generalization and interpretability, Materials Discovery 10 (2017) 19–28.

[26] X. Li, Y. Zhang, H. Zhao, C. Burkhart, L. C. Brinson, W. Chen, A transfer learning approach for microstructure reconstruction and structure-property predictions, Scientific reports 8 (1) (2018) 1–13.

[27] L. Mosser, O. Dubrule, M. J. Blunt, Reconstruction of three-dimensional porous media using generative adversarial neural networks, Physical Review E 96 (4) (2017) 043309.

[28] R. Cang, M. Y. Ren, Deep network-based feature extraction and reconstruction of complex material microstructures, in: ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, 2016, pp. V02BT03A008–V02BT03A008.

[29] R. Cang, Y. Xu, S. Chen, Y. Liu, Y. Jiao, M. Y. Ren, Microstructure representation and reconstruction of heterogeneous materials via deep belief network for computational material design, Journal of Mechanical Design 139 (7) (2017) 071404.

[30] R. Bostanabad, A. T. Bui, W. Xie, D. W. Apley, W. Chen, Stochastic microstructure characterization and reconstruction via supervised learning, Acta Materialia 103 (2016) 89–102.

[31] A. Z. Zinchenko, Algorithm for random close packing of spheres with periodic boundary conditions, Journal of Computational Physics 114 (2) (1994) 298–307.

[32] M. Groeber, M. Uchic, D. Dimiduk, Y. Bhandari, S. Ghosh, A framework for automated 3d microstructure analysis & representation, Journal of Computer-Aided Materials Design 14 (1) (2007) 63–74.

[33] M. Groeber, S. Ghosh, M. D. Uchic, D. M. Dimiduk, A framework for automated analysis and simulation of 3D polycrystalline microstructures. Part 1: Statistical characterization, Acta Materialia 56 (6) (2008) 1257–1273.

[34] M. Groeber, S. Ghosh, M. D. Uchic, D. M. Dimiduk, A framework for automated analysis and simulation of 3D polycrystalline microstructures. Part 2: Synthetic structure generation, Acta Materialia 56 (6) (2008) 1274–1287.

[35] D. T. Fullwood, S. R. Niezgoda, S. R. Kalidindi, Microstructure reconstructions from 2-point statistics using phase-recovery algorithms, Acta Materialia 56 (5) (2008) 942–948.

[36] D. Fullwood, S. Kalidindi, S. Niezgoda, A. Fast, N. Hampson, Gradient-based microstructure reconstructions from distributions using fast fourier transforms, Materials Science and Engineering: A 494 (1-2) (2008) 68–72.

[37] F. Latief, B. Biswal, U. Fauzi, R. Hilfer, Continuum reconstruction of the pore scale microstructure for fontainebleau sandstone, Physica A: Statistical Mechanics and its Applications 389 (8) (2010) 1607–1618.

[38] Y. Staraselski, A. Brahme, R. Mishra, K. Inal, Reconstruction of the 3D representative volume element from the generalized two-point correlation function, Modelling and Simulation in Materials Science and Engineering 23 (1) (2014) 015007.

[39] J. Feng, C. Li, S. Cen, D. Owen, Statistical reconstruction of two-phase random media, Computers & Structures 137 (2014) 78–92.

[40] D. Chen, X. He, Q. Teng, Z. Xu, Z. Li, Reconstruction of multiphase microstructure based on statistical descriptors, Physica A: Statistical Mechanics and its Applications 415 (2014) 240–250.

[41] H. Xu, D. A. Dikin, C. Burkhart, W. Chen, Descriptor-based methodology for statistical characterization and 3d reconstruction of microstructural materials, Computational Materials Science 85 (2014) 206–216.

[42] H. Xu, R. Liu, A. Choudhary, W. Chen, A machine learning-based design representation method for designing heterogeneous microstructures, Journal of Mechanical Design 137 (5) (2015) 051403.

[43] S. Chen, A. Kirubanandham, N. Chawla, Y. Jiao, Stochastic multi-scale reconstruction of 3d microstructure consisting of polycrystalline grains and second-phase particles from 2d micrographs, Metallurgical and Materials Transactions A (2016) 1440–1450.

[44] D. Li, M. A. Tschopp, M. Khaleel, X. Sun, Comparison of reconstructed spatial microstructure images using different statistical descriptors, Computational Materials Science 51 (1) (2012) 437–444.

[45] D. Li, X. Sun, M. Khaleel, Comparison of different upscaling methods for predicting thermal conductivity of complex heterogeneous materials system: application on nuclear waste forms, Metallurgical and Materials Transactions A 44 (1) (2013) 61–69.

[46] Z. Yang, Y. C. Yabansu, D. Jha, W.-k. Liao, A. N. Choudhary, S. R. Kalidindi, A. Agrawal, Establishing structure-property localization linkages for elastic deformation of three-dimensional high contrast composites using deep learning approaches, Acta Materialia 166 (2019) 335–345.

[47] Y. C. Yabansu, D. K. Patel, S. R. Kalidindi, Calibrated localization relationships for elastic response of polycrystalline aggregates, Acta Materialia 81 (2014) 151–160.

[48] Y. C. Yabansu, S. R. Kalidindi, Representation and calibration of elastic localization kernels for a broad class of cubic polycrystals, Acta Materialia 94 (2015) 26–35.

[49] D. M. de Oca Zapiain, E. Popova, S. R. Kalidindi, Prediction of microscale plastic strain rate fields in two-phase composites subjected to an arbitrary macroscale strain rate using the materials knowledge system framework, Acta Materialia 141 (2017) 230–240.

[50] R. Liu, Y. C. Yabansu, A. Agrawal, S. R. Kalidindi, A. N. Choudhary, Machine learning approaches for elastic localization linkages in high-contrast composite materials, Integrating Materials and Manufacturing Innovation 4 (1) (2015) 13.

[51] R. Liu, Y. C. Yabansu, Z. Yang, A. N. Choudhary, S. R. Kalidindi, A. Agrawal, Context aware machine learning approaches for modeling elastic localization in three-dimensional composite microstructures, Integrating Materials and Manufacturing Innovation 6 (2) (2017) 160–171.

[52] Z. Yang, Y. C. Yabansu, R. Al-Bahrani, W.-k. Liao, A. N. Choudhary, S. R. Kalidindi, A. Agrawal, Deep learning approaches for mining structure-property linkages in high contrast composites from simulation datasets, Computational Materials Science 151 (2018) 278–287.

[53] Z. Yang, R. Al-Bahrani, A. C. Reid, S. Papanikolaou, S. R. Kalidindi, W.-k. Liao, A. Choudhary, A. Agrawal, Deep learning based domain knowledge integration for small datasets: Illustrative applications in materials informatics, in: 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, 2019, pp. 1–8.

[54] P. K. Diederik, M. Welling, Auto-encoding variational bayes, in: Proceedings of the International Conference on Learning Representations (ICLR), Vol. 1, 2014.

[55] C. Doersch, Tutorial on variational autoencoders, arXiv preprint arXiv:1606.05908.

[56] D. Charte, F. Charte, S. García, M. J. del Jesus, F. Herrera, A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines, Information Fusion 44 (2018) 78–96.

DISTRIBUTION

1	MS 1323	Anh Tran, 1463
1	MS 1318	Tim Wildey, 1463
1	MS 1323	Dan Z. Turner, 1463
1	MS 1411	Theron Rodgers, 1864
1	MS 1411	Hojun Lim, 1864
1	MS 0346	Joe Bishop, 1556
1	MS 1411	Veena Tikhare, 1864
1	MS 1327	Justin Newcomer, 1462
1	MS 1303	Brad Boyce, 1881
1	MS 0899	Technical Library, 9536 (electronic copy)

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.