Chapter 7
System and Design

Abstract System and design bring the techniques of mechanistic data science (MDS) together to solve problems. The previous chapters have demonstrated the steps of MDS, starting with data collection through deep learning. This chapter will provide a range of examples for different classes of problems, from daily life topics to detailed science and engineering research. The examples shown in this chapter will include Type 1 (data driven problems), Type 2 (mixed data and scientific knowledge), and Type 3 (known mathematical science principles with uncertain parameters).

Keywords System and design · Spine growth · Piano and guitar music · Additive manufacturing · Polymer matrix composites · Indentation · Landslides · Tire design · Antimicrobial surfaces

7.1 Introduction

The previous chapters of this book have focused on the individual steps of a mechanistic data science analysis:

- Chapter 2 showed data generation and collection methodology,
- Chapter 3 showed methods for regression and optimization,
- Chapter 4 emphasized the extraction of mechanistic features,
- Chapter 5 showed methods of knowledge driven dimension reduction, and
- Chapter 6 focused on deep learning for regression and classification.

The ultimate objective of performing all these individual steps is to combine them to be able to "do something". This could range from developing new products or making critical decisions. In this chapter, some of the highlights of the various MDS from the previous chapters will be shown and some finality will be applied to the

Supplementary Information: The online version of this chapter (https://doi.org/10.1007/978-3-030-87832-0_7) contains supplementary material, which is available to authorized users.

W. K. Liu et al., *Mechanistic Data Science for STEM Education and Applications*,
https://doi.org/10.1007/978-3-030-87832-0_7

problems. The examples shown in this chapter will include Type 1 (data driven problems), Type 2 (mixed data and scientific knowledge), and Type 3 (known mathematical science principles with uncertain parameters).

7.2 Piano to Guitar Musical Note Conversion (Type 3 General)

Chapter 4 demonstrated the analysis of a sound wave signal made by a single piano note (note A4 on the piano keyboard) using Fourier transform (FT) analysis and short-time Fourier transform (STFT) analysis. The STFT results enabled the raw signal to be reduced to a reduced order model to be constructed, consisting of a spring-mass-damper. This reduced order model greatly reduced the dimension of the data. In Chap. 6, this same STFT methodology was applied to eight notes from a piano and to eight notes from a guitar, and these notes were used to train a neural network to convert a piano note to a guitar note. The challenge going forward is to be able to "make" alternative forms of music by converting the piano music to the form of another instrument. This chapter will also demonstrate how to perform the same conversion using principal component analysis (PCA).

7.2.1 Mechanistic Data Science with a Spring Mass Damper System

A comprehensive deep learning neural network algorithm for converting piano sounds to guitar sounds is shown in Fig. 7.1. When this algorithm is used with larger datasets, it can be used to convert musical melodies (multiple notes) from one

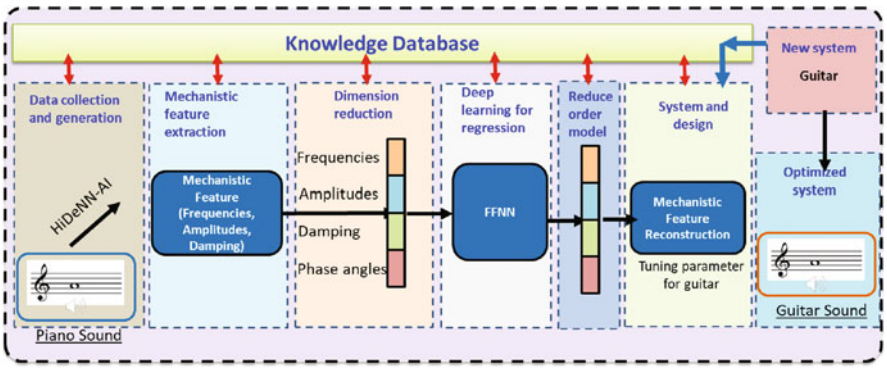

Fig. 7.1 Algorithm for converting from a piano to a guitar sound. Audio available in E-book (Supplementary Audio 7.1)

instrument to another. In this particular example, eight piano keys and eight guitar notes (A4, A5, B5, C5, C6, D5, E5, G5) are used for training a neural network for sound conversion (additional details are shown in Chap. 6).

The steps of the mechanistic data science methodology presented in this book are followed to achieve this musical sound conversion. The steps are outlined below. Note that some of the steps are grouped together because the lines between them are blurred and they are essentially executed in concert.

Step 1: Multimodal data collection and generation is the important first step in the process. Sound files for several different notes from both a piano and a guitar must be collected. For the analysis to be performed, 1.5–3 s of data for notes A4, A5, B5, C5, C6, D5, E5, G5 are collected. This dataset represents a minimum of data needed and additional data collection would enhance final results.

Representative sound waves are shown in Fig. 6.34 for a piano (left) and a guitar (right). As expected, the wave form for each is different. The amplitude for the piano decays much faster since there is a built-in damping mechanism. Difference such as this is manifested in the sound when each instrument is played. The piano has more reverberation whereas the guitar has a "twang" sound. Differences such as these lead to differences in the recorded sound files.

Steps 2, 3, and 4 of the MDS analysis process are all integrally intertwined and will be described together. These steps consist of extracting the mechanistic features of the sound signals and creating a reduced order surrogate model of each of the sound signals.

The key mechanistic features extracted for each of the sound signals are the (1) frequency (fundamental frequency and harmonic frequencies), (2) damping for each frequency, (3) phase angle, and (4) initial amplitude. An STFT is performed on each signal to separate the base signal into a set of frequencies composed of the fundamental frequency and the harmonic frequencies (Fig. 6.37).

Eight frequencies are extracted for each of the signals, with each frequency having a different damping, phase angle, and initial amplitude. A least squares optimization algorithm is used to determine the "best fit" properties for a spring-mass-damper mechanistic model for each of the component signals from the STFT of the original signals (Fig. 7.2).

This results in a reduced order model of the signals. As summarized in Table 7.1, the original signals were recorded with a frequency of 44,100 Hz, which lead to a dimension of 120,000 data points for the 2.8-s piano note recordings, and 72,000 data points for the 1.6-s guitar recordings. The mechanistic spring-mass-damper model only requires 4 features for the 8 frequencies extracted using the STFT, resulting in a dimension of 4 *features* x 8 *frequencies* $= 32$.

The reduced order model can be used to generate an approximation to the original sound file. Figure 7.3 shows the overlaid comparison of the two signals. It can be seen that reduced order model provides an accurate comparison, although late in the signal the comparison is not as close. The actual signal appears to increase in amplitude after 1.25 s, possibly due to reverberation or changing in the pressure from the damping pedal. Additional mechanistic features would be required to capture those effects.

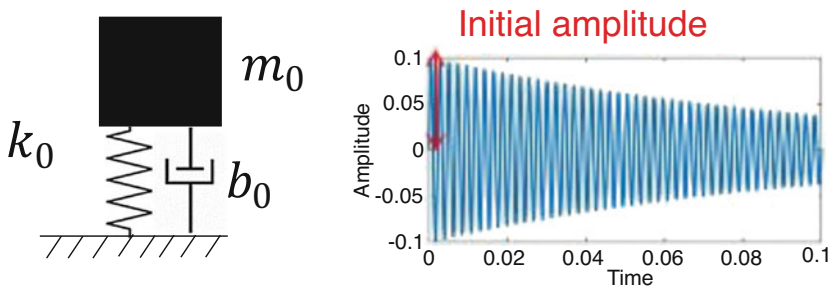

Fig. 7.2 Spring mass damper model (left) and representative wave form (right)

Table 7.1 Summary of dimension reduction achieved for the piano and guitar signals using the reduced order spring-mass-damper model

Instrument	Original signal dimension	Reduced order dimension
Piano	120,000 (44.1 kHz × 2.8 s)	32 (4 features × 8 frequencies)
Guitar	72,000 (44.1 kHz × 1.6 s)	32 (4 features × 8 frequencies)

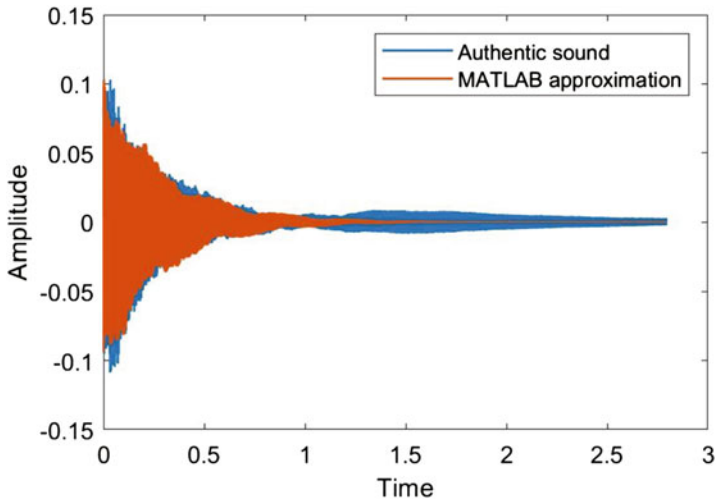

Fig. 7.3 Comparison of original authentic piano sound wave signal and a reduced order Matlab approximation. Audio available in E-book (Supplementary Audio 7.2)

Step 5: Deep learning for regression. The reduced order surrogate models for the piano notes and the guitar notes can be used to train a neural network to convert the sound from a given piano note to the corresponding guitar note. The four features (frequency, damping, phase angle, and amplitude) for the piano are used as input to a neural network and the weights and biases of the neural network are optimized to achieve the same four features that correspond to a guitar (Fig. 7.4).

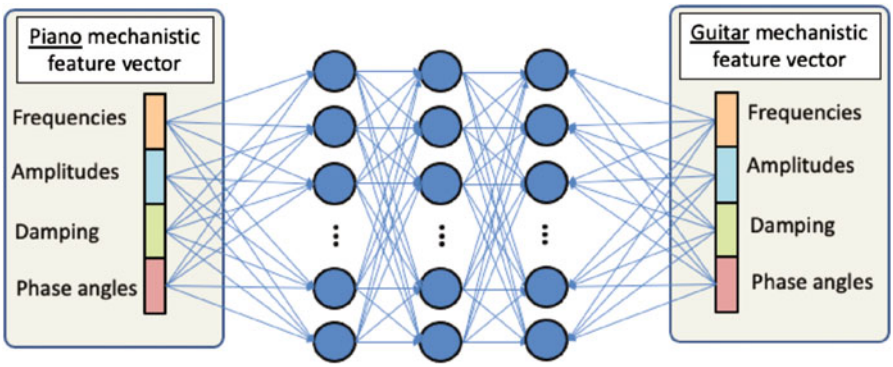

Fig. 7.4 Schematic of the neural network used to convert piano sounds to guitar sounds

This process is repeated for all the notes in the training dataset (A4, A5, B5, C5, C6, D5, E5, G5). Additional details of this process are given in Chap. 6.

Step 6: System and design.

A musical melody is a collection of notes played in sequence to make an enjoyable sound. For example, the blue signal at the top of Fig. 7.5 shows the first seven notes from the children's song "Twinkle, twinkle little star", played using piano notes. (For this example, the melody is made up of a collection of individually recorded notes in placed in sequence, but this exercise can also be performed for a melody played in one sitting on a piano.)

The goal of this example is to convert a melody from an authentic piano to a guitar sound using a trained neural network. To do this, the trained neural network described above and in Chap. 6 is used. The neural network is trained using eight piano notes and eight guitar notes as described earlier. As with all neural networks, training with more data yields enhanced results.

A reduced order model for each piano note in the melody is created by extracting the features with a STFT and using an optimization algorithm to fit the parameters. The features used are the frequency, damping, phase angle, and initial amplitude. The reduced order models of the piano notes are used as input to the trained neural network. The resulting output is the same note played using a guitar sound.

The first seven notes from "Twinkle, twinkle little star" are shown in Fig. 7.5. The blue signal (top) is made using the notes from an actual piano, and the green signal (middle) is made using the notes from an actual guitar. The orange signal at the bottom is created using the feed-forward neural network to convert the piano signal to a guitar sound. It can be seen from inspection of these signals that the authentic piano signal is quite different from the authentic guitar signal. In particular, the piano has a built-in damper that quickly reduces the amplitude of the signal. This accounts for the rapid decay of the piano signal. This characteristic is not present in a guitar and the green authentic guitar signal shows that the signal decay is much slower than the piano. The neural network, even trained using only eight piano notes, is able to capture this difference.

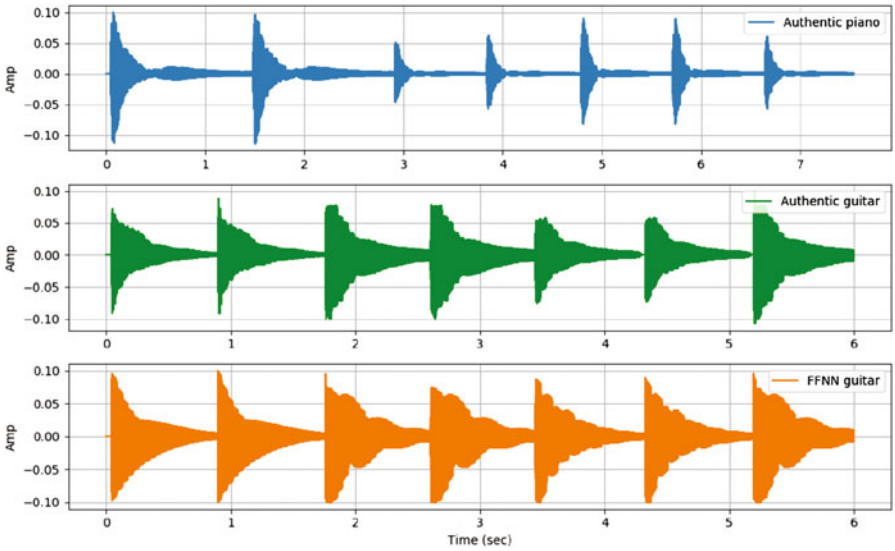

Fig. 7.5 First seven notes from "Twinkle, twinkle little star" from a piano (blue), from a guitar (green), and from a neural network trained using piano and guitar notes. Audio available in E-book (Supplementary Audio 7.3)

The Matlab and Python codes used for this example are given below (Supplementary File 7.1):

- Feature_extractor.m
- Sound_generator.m
- Model_trainer.py
- Feature_generator.py

Feature_extractor.m

```
clear all
clc
%% Read sound file

Filename='A4.wav';
Filename_out='A4.mat';
[y,Fs] = audioread(Filename);
%% STFT to find the damping coefficients, initial amplitudes, and the
frequencies
y=y(:,1);
t=0+1/Fs:1/Fs:1/Fs*size(y);
```

(continued)

```
% plot(t,y,"linewidth",1.5);
% xlim([0,0.01])
% xlabel('Time')
% ylabel('Amplitude')
x=y;
% Assumes x is an N x 1 column vector
% and Fs is the sampling rate.
N = size(x,1);
dt = 1/Fs;
t = dt*(0:N-1)';
dF = Fs/N;
f = dF*(0:N/2-1)';
X = fft(x)/N;
X = X(1:N/2);
X(2:end) = 2*X(2:end);
% figure;
% plot(f,abs(X),'linewidth',1.5);
% plot(f(1:100,1),abs(X(1:100,1)),'linewidth',1.5);
% xlim([0,2000]);
%
% xlabel('Frequency')
% ylabel('Amplitude')
X=abs(X);
[~,Index]=max(X);
basic_f=round(f(Index));
TF = islocalmax(X,'MinSeparation',basic_f.*1.9);
temp=f(TF);
omega=round(temp(1:8));
% figure
[s,f,t]=stft(y,Fs,'Window',  rectwin
(2048*2),'FFTLength',2048*2,'FrequencyRange','onesided');
% logs=abs(s(1:300,:));
% surface=surf(t,f(1:300,1),logs,'FaceAlpha',1);
% surface.EdgeColor = 'none';
% view(0,90)
% xlabel('Time(s)')
% ylabel('Frequency(Hz)')
% colormap(jet)
%temp=[452,904,1356,1808,2260,2737];
for i=1:1:8
  [~,Index(i)] = min(abs(f-omega(i)));
  amp=abs(s(Index(i),:));
  g = fittype('a*exp(b*x)');
  fit_f=fit(t,amp',g);
  out=coeffvalues(fit_f);
  a_out(i)=out(1);
  b_out(i)=out(2);
end
%% leasqure regression to find the optimal damping coeffcients,
initial amplitudes, phase angles
```

(continued)

```
[y,Fs] = audioread(Filename);
y=y(:,1);
t=0+1/Fs:1/Fs:1/Fs*size(y);

F = @(x,xdata)x(1).*exp(x(2).*xdata).*sin(2.*pi.*omega(1).
*xdata+x(3).*pi)+x(4).*exp(x(5).*xdata).*sin(2.*pi.*omega(2).
*xdata+x(6).*pi)+x(7).*exp(x(8).*xdata).*sin(2.*pi.*omega(3).
*xdata+x(9).*pi)+x(10).*exp(x(11).*xdata).*sin(2.*pi.*omega
(4).*xdata+x(12).*pi)+x(13).*exp(x(14).*xdata).*sin(2.*pi.
*omega(5).*xdata+x(15).*pi)+x(16).*exp(x(17).*xdata).*sin(2.
*pi.*omega(6).*xdata+x(18).*pi)+x(19).*exp(x(20).*xdata).*sin
(2.*pi.*omega(7).*xdata+x(21).*pi);
for i=1:1:8
  x0((i-1)*3+1)=a_out(i)/max(max(a_out)).*max(max(y));
  x0((i-1)*3+2)=b_out(i);
  x0((i-1)*3+3)=rand();
end % give the initial value of amplitude, damping coefficient and
phase angle
options = optimoptions(@fminunc,'Display','iter');
[x,resnorm,~,exitflag,output] = lsqcurvefit(F,x0,t,y',-10,10,
options);
for i=1:1:8
  a(i)=x0((i-1)*3+1);
  b(i)=x0((i-1)*3+2);
  phi(i)=x0((i-1)*3+3);
end
%% Save data
save(Filename_out,'a','b','phi','omega');
```

Sound_generator.m

```
clear all
clc
%% load original sound file and extracted features (original sound
file is just used to measure the duration)
Filename='A4.wav';
Filename_data='A4.mat';
Filename_sound_file='A4_MATLAB.wav';
[y,Fs] = audioread(Filename);
load(Filename_data)
amp_factor=0.1087; %sound volume normalization factor
%% generate sound single

y=y(:,1);
t=0+1/Fs:1/Fs:1/Fs*size(y);

y_t=zeros(size(t));
for i=1:1:7
```

(continued)

```
 y_t=y_t+a(i).*exp(b(i).*t).*sin(2.*pi.*omega(i).*t+phi(i).
*pi);
end

y_t=y_t./max(abs(y_t)).*amp_factor;

plot(t,y,'linewidth',1);
hold on
plot(t,y_t,'linewidth',1);

legend("Authentic sound","MATLAB approximation")
xlabel('Time')
ylabel('Amplitude')

%% write sound file
audiowrite(Filename_sound_file,y_t,Fs);
```

Model_trainer.py

```
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
from torch.utils.data import DataLoader

from tools import *

def model_trainer(dataset_path):
  '''train model

  Args:
   dataset_path: [String] folder to save dataset, please name it as
"dataset";

  Returns:
    None, but save model to current_folder + "results/mode.pkl"
  '''
  # configuration
  config = Configer()

  dataset_train = MyDataset(dataset_path, 'train')
  dataset_test = MyDataset(dataset_path, 'test')
```

(continued)

```
  print(f'[DATASET] The number of paired data (train): {len
(dataset_train)}')
  print(f'[DATASET] The number of paired data (test): {len
(dataset_test)}')
  print(f'[DATASET] Piano_shape: {dataset_train[0][0].shape},
guitar_shape: {dataset_train[0][1].shape}')

  # dataset
  train_loader = DataLoader(dataset_train, batch_size=config.
batch_size, shuffle=True)
  test_loader = DataLoader(dataset_test, batch_size=config.
batch_size, shuffle=True)

  net = SimpleNet(config.p_length, config.g_length)
  net.to(device)
  criterion = nn.MSELoss()
  optimizer = optim.Adam(net.parameters(), lr=config.lr)
  scheduler = StepLR(optimizer, step_size=int(config.epoch/4.),
gamma=0.3)

  # Note that this part is about model_trainer
  loss_list = []
  for epoch_idx in range(config.epoch):
    # train
    for step, (piano_sound, guitar_sound, _) in enumerate
(train_loader):
      inputs = piano_sound.to(device)
      targets = guitar_sound.to(device)
      inputs = inputs.reshape(inputs.shape[0], 4, -1)
      targets = targets.reshape(inputs.shape[0], 4, -1)

      optimizer.zero_grad()
      outputs = net(inputs)
      loss = criterion(outputs, targets)
      loss_list.append(loss.item())
      loss.backward()
      optimizer.step()

    # eval
    if epoch_idx % int(config.epoch/10.) == 0:
      net.eval()
      for step, (inputs, targets, _) in enumerate(train_loader):
        inputs = inputs.to(device)
        targets = targets.to(device)
        inputs = inputs.reshape(inputs.shape[0], 4, -1)
        targets = targets.reshape(inputs.shape[0], 4, -1)
        outputs = net(inputs)
      loss = criterion(outputs, targets)
```

(continued)

```
    print(f'epoch: {epoch_idx}/{config.epoch}, loss: {loss.item
()}')

  # save model
  torch.save(net.state_dict(), dataset_path.replace('dataset',
'results')+'/model.pkl')

  # plot loss history
  fig = plt.figure()
  plt.plot(loss_list, 'k')
  plt.ylim([0, 0.02])
  plt.xlabel('Iteration', fontsize=16)
  plt.ylabel('Loss', fontsize=16)
  plt.tight_layout()
  plt.savefig('results/MDS_loss.jpg', doi=300)

if __name__ == '__main__':
  # train model
  dataset_path = 'dataset'
  model_trainer(dataset_path)
```

Guitar_feature_generator.py

```
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader

from tools import *

def guitar_feature_generator(dataset_path, key_name):
  '''Generate predicted guitar features from piano features

  Args:
   dataset_path: [String] folder to save dataset, please name it as
"dataset";
    key_name: [String] key name that you want to generate. Example:
"A4"

  Returns:
   gen_guitar_feats: [List] contains predicted guitar features in
a dict,
```

(continued)

```
            note that this part can be used to generate many guitar
features,
             so we use a list to store the guitar features.
  '''
  config = Configer()
  model_path = dataset_path.replace('dataset', 'results')+'/
model.pkl'
  net = SimpleNet(config.p_length, config.g_length)
  net.to(device)
  net.load_state_dict(torch.load(model_path))
  net.eval()

  res, res_true = [], []
  dataset_train = MyDataset(dataset_path, 'train')
  train_loader = DataLoader(dataset_train, batch_size=config.
batch_size, shuffle=True)
  for step, (inputs, targets, key_names) in enumerate
(train_loader):
    inputs = inputs.to(device)
    targets = targets.to(device)
    inputs = inputs.reshape(inputs.shape[0], 4, -1)
    targets = targets.reshape(inputs.shape[0], 4, -1)
    outputs = net(inputs)

    gen_feats_batch = outputs.detach().cpu().numpy()
    targets_batch = targets.detach().cpu().numpy()
    inputs_batch = inputs.detach().cpu().numpy()

    for i in range(gen_feats_batch.shape[0]):
      if key_names[i] != key_name:
        continue

      pred_feats_norm = gen_feats_batch[i].reshape(4,8)
      # inverse data to original range
     pred_feats = dataset_train.inverse_guitar(pred_feats_norm)
      true_feats_norm = targets_batch[i].reshape(4,8)
     true_feats = dataset_train.inverse_guitar(true_feats_norm)
      inputs_feats_norm = inputs_batch[i].reshape(4,8)
      inputs_feats = dataset_train.inverse_piano
(inputs_feats_norm)

      d = {
        'key': key_names[i],
        'freq': pred_feats[0,:],
        'phi': pred_feats[1,:],
        'a': pred_feats[2,:],
        'b': pred_feats[3,:],
      }
      d_true = {
```

(continued)

```
        'key': key_names[i],
        'freq': true_feats[0,:],
        'phi': true_feats[1,:],
        'a': true_feats[2,:],
        'b': true_feats[3,:],
      }
      res_true.append(d_true)
      res.append(d)

      # plot results
      fig = plt.figure(figsize=(12,5))
      ax1 = fig.add_subplot(1, 2, 1)
      lns1 = plt.plot(pred_feats[0,:], pred_feats[2,:], '^',
label='Prediction (G)')
      lns2 = plt.plot(true_feats[0,:], true_feats[2,:], 'v',
label='Ground Truth (G)')
      plt.xlabel('Frequency', fontsize=16)
      plt.ylabel('Amplitude', fontsize=16)
      ax2 = ax1.twinx()
      lns3 = plt.plot(inputs_feats[0,:], inputs_feats[2,:], 'o',
c='g', label='Ground Truth (P)')
      lns = lns1+lns2+lns3
      labs = [l.get_label() for l in lns]
      ax1.legend(lns, labs, loc=0, fontsize=14)
      plt.title('Key: '+key_names[i], fontsize=18)

      ax3 = fig.add_subplot(1, 2, 2)
      lns1 = plt.plot(pred_feats[1,:], pred_feats[3,:], '^',
label='Prediction (G)')
      lns2 = plt.plot(true_feats[1,:], true_feats[3,:], 'v',
label='Ground Truth (G)')
      plt.xlabel('Phase angle', fontsize=16)
      plt.ylabel('Dampping coefficient $b_i$', fontsize=16)
      ax4 = ax3.twinx()
      lns3 = plt.plot(inputs_feats[1,:], inputs_feats[3,:], 'o',
c='g', label='Ground Truth (P)')
      lns = lns1+lns2+lns3
      labs = [l.get_label() for l in lns]
      ax3.legend(lns, labs, loc=0, fontsize=14)
      plt.title('Key: '+key_names[i], fontsize=18)

      plt.tight_layout()
      plt.savefig(f'results/MDS_pred_{key_names[i]}.jpg',
doi=300)
  return res

if __name__ == '__main__':
  dataset_path = 'dataset'
  # generate guitar features from piano features
```

(continued)

```
  gen_guitar_feats = guitar_feature_generator(dataset_path,
'A4')
  print('gen_guitar_feats', gen_guitar_feats)

  # show prediction results
  for dt in gen_guitar_feats:
    for key, value in dt.items():
      print("{}={}'".format(key, list(value)))
```

7.2.2 *Principal Component Analysis for Musical Note Conversion (Type 1 Advanced)*

The dimensions of the raw sound signals can be reduced using principal component analysis (PCA). It should be noted that PCA creates a reduced order model based only on the data and does not consider mechanistic features, and thus it is very hard to interpret the meanings of the reduced model created by the PCA. The principal components of the data are extracted using the methodology described in Sect. 5.3.

Step 1. Dataset collection

The training data is the same as the mechanistic data science model. It consists of eight pairs of piano and guitar sound signals, including A4, A5, B5, C5, C6, D5, E5, and G5 notes for each instrument. Detailed information for the training set can be found in Sect. 6.4.2.

Step 2. Extraction of dominant features by PCA

7.2.3 *Data Preprocessing (Normalization and Scaling)*

Firstly, the piano sound signals are put into a matrix $\boldsymbol{A_p}$, in which each row represents a sound signal, and each column represents the amplitude at a certain time step. The dimension of $\boldsymbol{A_p}$ is m (8) $\times$ n (81, 849), where m is the number of piano sound files and n is the minimum number of time steps (duration $\times$ sample rate) in all piano sound files. The piano A5 key has the minimum number of time step (81, 849 $\approx$ 1.86 s $\times$ 44,100 Hz). Matrix $\boldsymbol{A_p}$ can be illustrated as (Fig. 7.6):

The mean and standard deviation are calculated for each time step of the matrix $\boldsymbol{A_p}$, defined as:

$$mean\left(\boldsymbol{A_p}\right) \text{and } std\left(\boldsymbol{A_p}\right).$$

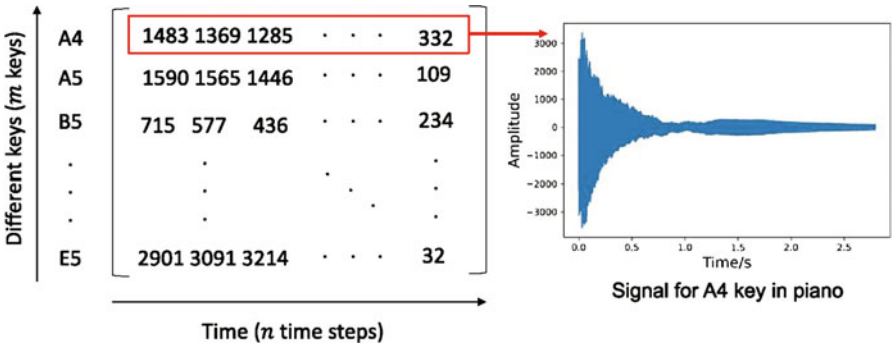

Fig. 7.6 Build a matrix $\boldsymbol{A_p}$ for all piano sounds. The dimension of $\boldsymbol{A_p}$ is $m \times n$

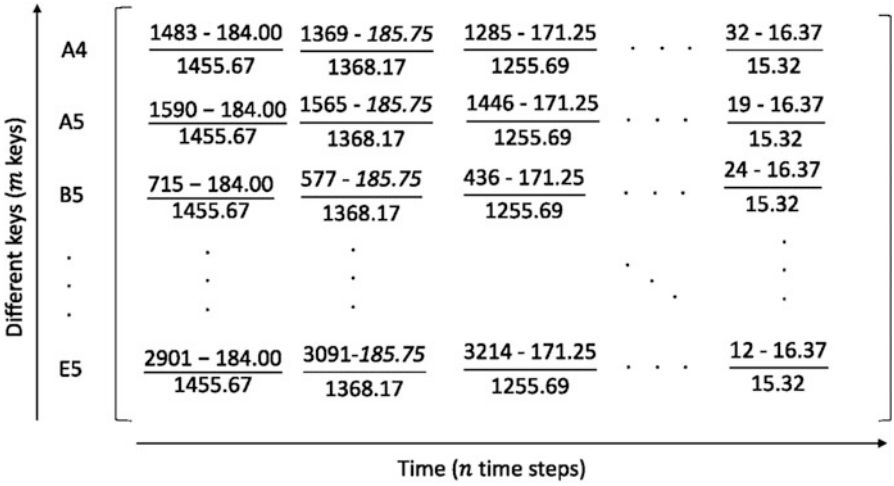

Fig. 7.7 Obtain a normalized and scaled matrix $\boldsymbol{B_p}$. The dimension of $\boldsymbol{B_p}$ is $m \times n$

The dimensions of $mean(\boldsymbol{A_p})$ and $std(\boldsymbol{A_p})$ are both $n \times 1$ matrices. The matrix $\boldsymbol{A_p}$ is then normalized column by column by subtracting the $mean(\boldsymbol{A_p})$ and dividing the $std(\boldsymbol{A_p})$. Note that in this case, $\boldsymbol{A_p}$ is scaled by dividing by the standard deviation to accelerate the training of the fully-connected FFNN. The normalized and scaled matrix is called $\boldsymbol{B_p}$, and has a dimension the same as $\boldsymbol{A_p}$.

The same procedure is used to obtain a normalized and scaled matrix $\boldsymbol{B_g}$, using mean vector $mean(\boldsymbol{A_g})$ and standard vector $std(\boldsymbol{A_g})$, for the guitar sound signals (Fig. 7.7).

7.2.4 Compute the Eigenvalues and Eigenvectors for the Covariance Matrix of B_p and B_g

The covariance matrix, X_p, of normalized matrix B_p is calculated as:

$$X_p = \frac{B_p^T B_p}{n-1} \tag{7.1}$$

A Singular Value Decomposition (SVD) is then performed for the covariance matrix, X_p, (additional details in Sect. 5.4):

$$X_p = P_p \Lambda P_p^T = \begin{bmatrix} p_1^p & p_2^p & \dots & p_m^p \end{bmatrix} \begin{bmatrix} \lambda_1^p & 0 & \dots & 0 \\ 0 & \lambda_2^p & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \lambda_m^p \end{bmatrix} \begin{bmatrix} p_1^{pT} \\ p_2^{pT} \\ \dots \\ p_m^{pT} \end{bmatrix} \tag{7.2}$$

where P_p is an orthogonal matrix containing the eigenvectors ($p_1^p, p_2^p, p_3^p, \dots, p_m^p$) and Λ is a diagonal matrix containing the eigenvalues ($\lambda_1^p, \lambda_2^p, \lambda_3^p, \dots, \lambda_m^p$).

7.2.5 Build a Reduced-Order Model

The normalized piano sounds matrix, B_p, can be projected to the eigenvectors to obtain a reduced-order model, R_p:

$$R_p = B_p P_p \tag{7.3}$$

As the dimension of B_p is $m \times n$ and the dimension of P_p is $n \times m$, the dimension of R_p is $m \times m$.

The ith row in R_p represents a reduced-dimension vector a_i for a piano sound:

$$R_p = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix} \tag{7.4}$$

The vector a_i contains the magnitudes of all principal components (PCs) for the ith piano sound. The dimension of a_i is $m \times 1$.

The same procedure can be followed for the guitar sound signals. The ith row in R_g represents a reduced-dimension vector b_i for a guitar sound:

Table 7.2 Eight principal components for A4 key in piano and guitar

Magnitude for each PC	Piano-A4	Guitar-A4
1st PC	−78.08	−47.32
2nd PC	303.93	−3.18
3rd PC	47.98	19.68
4th PC	−5.03	27.26
5th PC	−38.21	−40.95
6th PC	−5.68	19.26
7th PC	−4.71	144.83
8th PC	1.45×10^{-13}	3.58×10^{-14}

$$\boldsymbol{R_g} = \begin{bmatrix} \boldsymbol{b_1^T} \\ \boldsymbol{b_2^T} \\ \vdots \\ \boldsymbol{b_m^T} \end{bmatrix} \tag{7.5}$$

For the A4 key, the magnitudes of eights PCs for the piano and guitar sound are shown in Table 7.2. A detailed explanation of PCA can be found in Sect. 5.3.

7.2.6 Inverse Transform Magnitudes for all PCs to a Sound

Using the vector $\boldsymbol{b_i}$ that contains all magnitudes for each guitar PC, $\boldsymbol{b_i}$ can be inversely transformed to a guitar sound, $\boldsymbol{s_i}$, by multiplying by $\boldsymbol{P_p^T}$, doing an element-wise product of $std(\boldsymbol{A_g})$, and adding the $mean(\boldsymbol{A_g})$:

$$\boldsymbol{s_i} = \boldsymbol{b_i^T P_g^T} \circ std\left(\boldsymbol{A_g}\right) + mean\left(\boldsymbol{A_g}\right) \tag{7.6}$$

Note that $\circ$ is a notation for an element-wise product.

7.2.7 Cumulative Energy for Each PC

One important part of using PCA in practice is to estimate the number of PCs needed to describe the data. For the piano sounds, the cumulative energy, e_i, of ith PC can be defined as:

$$e_i = \sum_{k=1}^{i} \left(\lambda_k^p\right)^2 / \sum_{k=1}^{m} \left(\lambda_k^p\right)^2$$

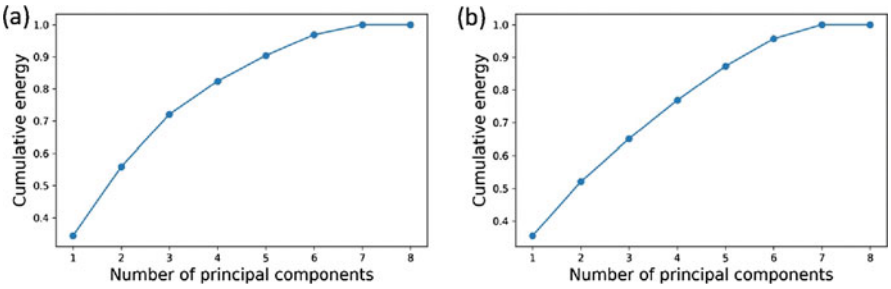

Fig. 7.8 The relationship between cumulative energy and the number of principal components. (a) Piano sounds. (b) Guitar sounds

The cumulative energy for each PC is shown in Fig. 7.8. Using only 5 principal components captures 90% of the information of the original sounds.

7.2.8 Python Code for Step 1 and Step 2

The Python code for data collection and data preprocessing is shown below:

```
def fit_norm_PCA(self, itype, min_length, data_type):
 '''Fit normalization and PCA scaler for training set
 Args:
    itype: [string] decide whether you are handling piano or guitar
sounds
    min_length: [int] the minimum length for sounds files
    data_type: [string] train or test
  Returns:
    PCA_scaler: [object] a PCA scaler
    mean: [array] mean for this instrument
    std: [array] standard deviation for this instrument
 '''
 # load all sound files for a certain instrument
 sound_file_all = glob.glob(f'{dataset_path}/{itype}/train/*.
wav')
 # load wav files to a list
 sounds_all = self.load_files(sound_file_all)
 # segment all sounds with the minimum length and convert to an array
  sounds_seg = np.array([i[:min_length] for i in sounds_all])

 # normalization and scaling
 mean = np.mean(sounds_seg, 0)
```

(continued)

```
std = np.std(sounds_seg, 0)
 sounds_seg_norm = (sounds_seg - mean) / std

 # fit PCA scaler
 PCA_scaler = PCA(n_components=self.n_components)
 PCA_scaler.fit(sounds_seg_norm)

 return PCA_scaler, mean, std
```

Step 3. Deep learning regression

7.2.9 *Training a Fully-Connected FFNN*

After obtaining eight piano vectors $\boldsymbol{a}_{1,2,3,\ldots,8}$ and eight guitar vectors $\boldsymbol{b}_{1,2,3,\ldots,8}$, a same fully-connected FFNN model, $\mathcal{F}_{FFNN}$, is built to learn the mapping between piano sounds and guitar sounds. The input is a vector $\boldsymbol{a}_i$ that contains the magnitudes of PCs of ith piano sound. The output is a predicted vector $\widehat{\boldsymbol{b}_i}$ that contains the magnitudes of PCs of ith guitar sound. The FFNN can be defined as:

$$\widehat{\boldsymbol{b}_i} = \mathcal{F}_{FFNN}(\boldsymbol{a}_i) \tag{7.7}$$

The FFNN has three hidden layers with 100 neurons. Each hidden layer uses a *tanh* function as the activation function to learn the non-linear relationship. A detailed description of the FFNN can be found in Sects. 6.2.2 and 6.4.2. The only difference between the mechanistic data science in Sect. 6.4.2 and PCA-NN is that the dimension of input and output are not the same. The structure of the PCA-NN is shown in Fig. 7.9.

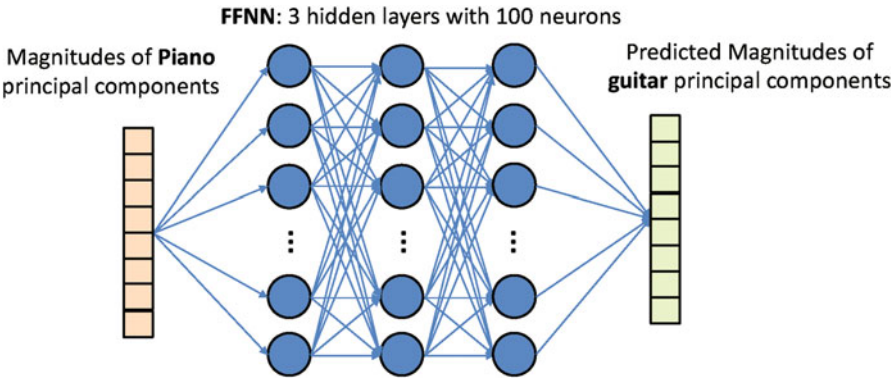

Fig. 7.9 FFNN structure for PCA-NN model. The input and output are principal components, which are reduced dimension features

The loss function $\mathcal{L}$ calculates the Mean Squared Error (MSE) between authentic magnitudes of guitar principal components $\boldsymbol{b_i}$ and predicted magnitudes of guitar principal components $\widehat{\boldsymbol{b_i}}$:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{b_i} - \widehat{\boldsymbol{b_i}} \right)^2 \tag{7.8}$$

During training, the FFNN is fed the same notes for the different instruments. For example, $\boldsymbol{a_1}$ can be fed into the FFNN as the input and $\boldsymbol{b_1}$as the output. The subscript 1 indicates that the vectors are for the A4 key. Then, the FFNN is trained for another paired set of vectors $\boldsymbol{a_2}$ and $\boldsymbol{b_2}$, corresponding to the A5 key. The FFNN is iteratively trained until it reaches the maximum iteration.

7.2.10 Code Explanation for Step 3

PyTorch is used to implement the FFNN and to train the model:

```
def model_trainer(dataset_path):
  '''train a FFNN model
  Args:    dataset_path: [string] folder to save dataset, please
name it as "dataset";
  Returns:    None, but save a trained model  '''
  # set your hyperparameters
  config = Configer()
  # load dataset
 dataset_train = MyDataset(dataset_path, 'train', config.
n_components)
  # build dataloader
  train_loader = DataLoader(
    dataset_train, batch_size=config.batch_size, shuffle=True)
  # initialize the FFNN
  net = SimpleNet(input_dim=config.n_components,
          output_dim=config.n_components)
 net.to(device)
  # build a MSE loss function
 criterion = nn.MSELoss()
 # Use adam as the optimizer
 optimizer = optim.Adam(net.parameters(), lr=config.lr)
 # step learning rate decay
  scheduler = StepLR(optimizer, step_size=int(config.epoch/4.),
gamma=0.3)
 loss_list = []  # to save the loss
 # iterate all epoches
```

(continued)

```
 for epoch_idx in range(config.epoch):
    for step, (piano_sound, guitar_sound, _) in enumerate
(train_loader):
      # data preparation for PyTorch
   inputs = piano_sound.to(device)
      targets = guitar_sound.to(device)
      inputs = inputs.reshape(inputs.shape[0], config.
n_components)
      targets = targets.reshape(inputs.shape[0], config.
n_components)
  # backpropagation to update parameters of the FFNN
      optimizer.zero_grad()
      outputs = net(inputs)
      loss = criterion(outputs, targets)
      loss_list.append(loss.item())
      loss.backward()
      optimizer.step()
```

Step 4. Generate a single guitar sound

7.2.11 Generate a Single Guitar

After training, a well-trained FFNN model $\mathcal{F}_{FFNN}$ is obtained that can be used to predict the magnitudes of guitar PCs $\widehat{\boldsymbol{b}}$ using the magnitudes of a piano sound PCs $\boldsymbol{a}$ as input:

$$\widehat{\boldsymbol{b}} = \mathcal{F}_{FFNN}(\boldsymbol{a}) \tag{7.9}$$

In this example, the vector $\boldsymbol{a}$ comes from the training set. Then, the predicted guitar PCs $\widehat{\boldsymbol{b}}$ is inversely transformed to a guitar sound $\widehat{\boldsymbol{s_g}}$ by projecting it to the original space:

$$\widehat{\boldsymbol{s_g}} = \widehat{\boldsymbol{b}}^T \boldsymbol{P_g^T} \circ std(\boldsymbol{A_g}) + mean(\boldsymbol{A_g}) \tag{7.10}$$

The reconstruction results are shown in Fig. 7.10. The PCA-NN generated guitar sound wave is almost the same as the authentic one. Figure 7.10 demonstrates that the PCA-NN model captures the key features compared to the authentic guitar sound.

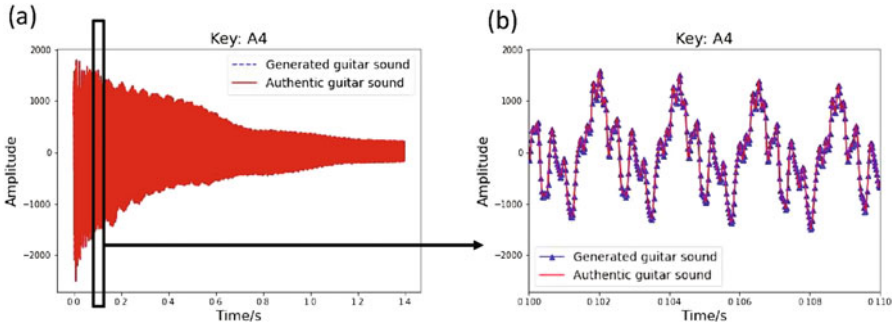

Fig. 7.10 Time-amplitude curves (sound waves) of reconstructing a single guitar key A4 from a piano key's PCs as input. (**a**) PCA-NN generated guitar sound. (**b**) Magnification plot of PCA-NN generated guitar sound ranging from 0.10 to 0.11 s, to highlight the detailed wave shape

7.2.12 Python Code for Step 4

Inverse transform Python code is shown below:

```
def inverse(self, itype, dt):
 '''inverse data to original scale
 Args:
    itype: [string] decide whether you are handling piano or guitar
sound
     dt: [array] the magnitudes for guitar PCs. The length is the
number of
        principal components
 Returns:
     Sound_raw: [array] an array that contains the signal in the
original
                space
 '''
 if itype == 'piano':
   # inverse transform for PCA
    sound_norm = self.piano_PCA_scaler.inverse_transform(dt)
    # inverse transform for normalization
    sound_raw = sound_norm * self.piano_std + self.piano_mean
 elif itype == 'guitar':
   # inverse transform for PCA
    sound_norm = self.guitar_PCA_scaler.inverse_transform(dt)
    # inverse transform for normalization
    sound_raw = sound_norm * self.guitar_std + self.guitar_mean

  return sound_raw
```

Step 5. Generate a melody

7.2.13 Generate a Melody

To generate a melody for part of the children's song *Twinkle, Twinkle Little Star*, step 3 is followed to prepare all related sound files, including A5, C5, and G5. Then, all notes are combined in the proper order, based on the order of the notes in the melody, i.e., "C5, C5, G5, G5, A5, A5, G5".

7.2.14 Code Explanation for Step 5

The generation of a melody can be performed by a command-line code using FFmpeg, a popular open-source audio and video software:

```
ffmpeg -i C5.wav -i C5.wav \
-i G5.wav -i G5.wav \
-i A5.wav -i A5.wav -i G5.wav \
-filter_complex '[0:0][1:0][2:0][3:0][4:0][5:0][6:0]
concat=n=7:v=0:a=1[out]' \
-map '[out]' melody.wav
```

7.2.15 Application for Forensic Engineering

Forensic engineering involves the application of mathematical science and engineering methods to analyze "clues" that are left following some "event". This "event" may involve an accident or an alleged failure, but it can also consist of analyzing product performance to determine if it is working properly.

While one may not think of a musical instrument in this context, a hypothetical scenario can be constructed that would represent a typical forensic engineering example. Consider a situation in which someone pays a large amount of money for a special collector's item piano supposedly owned by a famous musician. After getting it to their home, the piano does not sound right. To better understand the piano, musical notes from the piano in its current condition can be compared with previous recordings from the piano.

This hypothetical scenario may not be very common, but this same methodology can be used to classify noises or other from mechanical systems. Engineers regularly record vibrations and sounds on automobiles and other machinery to study squeaks

and rattles. Based on the frequencies and other signal characteristics, problems with brakes, bearings, or engine components can be identified and classified.

7.3 Feature-Based Diamond Pricing (Type 1 General)

Predicting the price of a diamond based on its features, such as color, clarity, cut, carat weight is a type 1 problem for mechanistic data science (see Chap. 1). In the book, this diamond pricing example is used in several sections to explain key concepts of the data science. Starting from a large repository of data (Chap. 2) on diamond features and prices with appropriate data normalizations (Chap. 4), the diamonds can be classified into multiple clusters in which the diamonds have similar features and prices. Many clustering methods can achieve this goal, for example, the k-means clustering that introduced in Chap. 5. Mapping the diamonds' features to their prices, regression methods can be applied such as linear regression (Chap. 3), nonlinear regression (Chap. 3), and neural networks (Chap. 6). Finally, given a new diamond with known features, its price can be predicted by the trained machine learning model. This model can help customers find a reasonable price for the diamond they would like to purchase, and also help diamond sellers evaluate the value of diamonds.

7.4 Additive Manufacturing (Type 1 Advanced)

Additive manufacturing (AM), sometimes called three-dimensional (3D) printing, is a rapidly growing advanced manufacturing paradigm that promises unparalleled flexibility in the production of metal or non-metal parts with complex geometries. However, the nature of the process creates position-dependent microstructures, defects, and mechanical properties that complicate printing process design, part qualification, and manufacturing certification. Metal additive manufacturing, such as laser powder bed fusion (L-PBF) and directed energy deposition (DED), have most of the relevant physical processes occurring in the vicinity of the melt pool. The laser rapidly heats the metal causing localized melting and vaporization. The melt pool surface extends behind the moving laser producing large thermal gradients with corresponding variations in surface tension. During rapid solidification, microstructure growth can produce complicate phases and grain morphologies that strongly affect the local component properties and performances. These multiscale and multiphysics phenomena involve interactions and dependencies of a large number of process parameters and material properties leading to complex process-structure-properties (PSP) relationships. In AM, localized heating/cooling heterogeneity leads to spatial variations of as-built mechanical properties, significantly complicating the materials design process. To this end, a mechanistic data-driven framework [1] integrating wavelet transforms and convolutional neural networks is developed to

predict position-dependent mechanical properties over fabricated parts based on process-induced temperature sequences, i.e., thermal histories. The in-situ thermal histories were measured by a well-calibrated infrared (IR) camera during DED process of Inconel 718 material. The mechanical properties of interest include ultimate tensile strength (UTS), yield strength, and elongation. The framework enables multiresolution analysis to reveal dominant mechanistic features underlying the additive manufacturing process, such as fundamental thermal frequencies. The approach provides a concrete foundation for a revolutionary methodology that predicts spatial and temporal evolution of mechanical properties leveraging domain-specific knowledge and cutting-edge machine and deep learning technologies. The proposed data-driven supervised learning approach is aimed to capture nonlinear

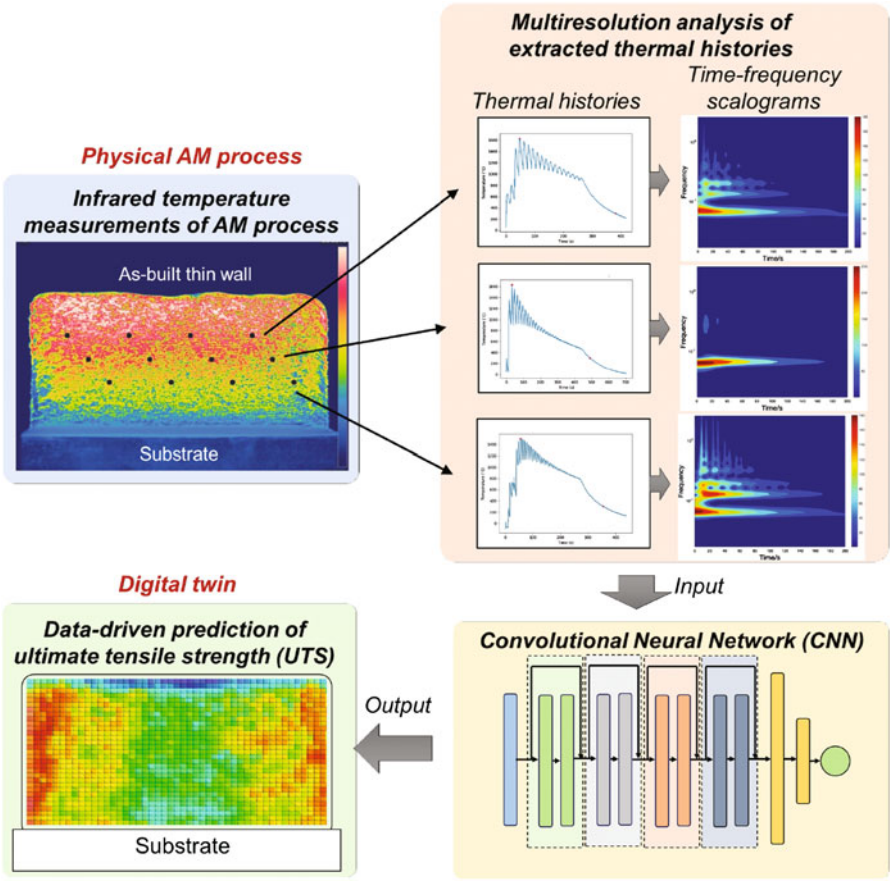

Fig. 7.11 A schematic of the proposed mechanistic data-driven model linking thermal history and mechanical properties such as ultimate tensile strength (UTS). A convolutional neural network (CNN) scheme combing with a multiresolution analysis is developed to deal with high-dimensional thermal histories and a small amount of noisy experimental data. This methodology provides a mechanistic data-driven framework as a digital twin of physical AM process [1]

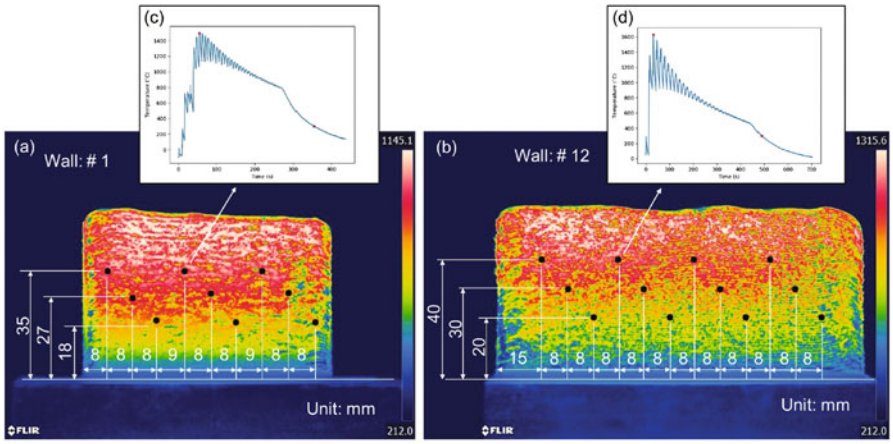

Fig. 7.12 Illustrative IR measurements. (**a**) Positions of areas of interest at wall #1. (**b**) Positions of areas of interest at wall # 12. (**c**) An illustrative thermal history for a specific area at wall # 1. (**d**) An illustrative thermal history for a specific area at wall # 12. The part numbers are provided in the reference [1]

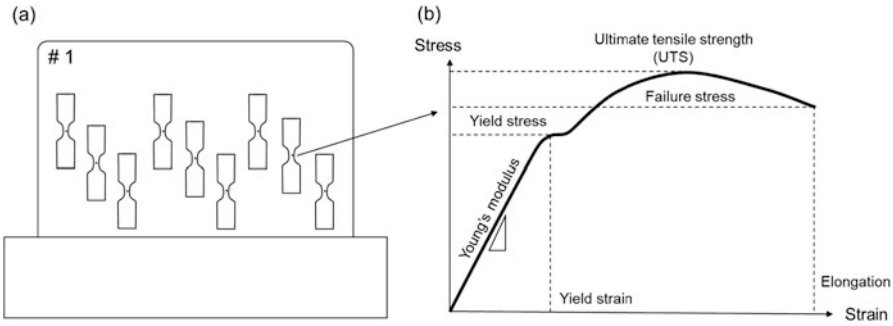

Fig. 7.13 Illustrative tensile specimens cut from different area of a wall and the stress-strain curve. (**a**) Positions of tensile specimens at wall # 1. (**b**) A schematic of stress-strain curve and corresponding mechanical properties

mapping between local thermal histories (i.e., time-temperature curves) and as-built mechanical properties such as UTS, which is defined as the ability of a material to resist a force that tends to pull it apart. A schematic of the proposed framework is shown in Fig. 7.11.

Step 1: Multimodal data generation and collection

Twelve sets of thermal history were collected by IR *in-situ* measurement for 12 additively manufactured thin walls (5000 uniformly spaced measurement locations per wall). Each thin wall was built using a single track and multilayer laser DED process. For mechanical tensile tests, 135 specimens were cut at

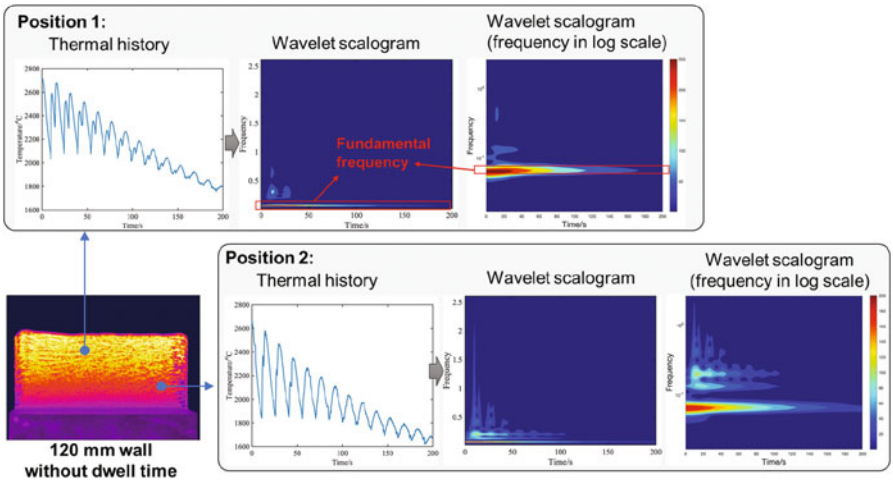

Fig. 7.14 A schematic showing the local thermal history and corresponding wavelet scalograms at different locations on the as-built wall when there is no dwell time

specific positions of interest. For training the proposed data-driven model, 135 sets of thermal history were used as input (Fig. 7.12) and corresponding 135 sets of UTS were used as labeled output (Fig. 7.13). Detailed information for thermal history extraction and mechanical tensile tests is provided in the reference[1]. All the thermal histories (5000 per wall) were then used as input of trained data-driven model to predict 2D high-resolution UTS maps for each thin wall fabricated by a specific process condition.

Steps 2, 3 and 4: Extraction of mechanistic features, knowledge-driven dimension reduction, and reduced order surrogate models.

To extract mechanistic meanings of the thermal histories and improve the predictive capability of the model given a small amount of noisy data, the high-dimensional thermal histories are transformed into time-frequency spectrums using wavelet transforms [2]. Feature engineering is performed by applying wavelet analysis on the experimental time-temperature histories (i.e., thermal histories). Underlying mechanistic information can be revealed by wavelet transformed time-frequency maps. Figure 7.14 shows an example where the wall without dwell time was considered. The time-temperature histories at different points were converted into time-frequency maps using wavelet transform. Thermal histories vary depending on the position on the wall. For example, at the top-left of the wall (position 1 in the Fig. 7.14), the thermal history shows dual peaks. This happens because the specimen point is slight to the left, and the point does not get sufficient cooling time before being reheated again. The heating-cooling cycles are manifested as periodic in nature in the time-temperature history. The time-temperature history from the bottom-right of the wall (position 2) has a shorter period for heating-reheating as the point is near the corner. If we compare the wavelet transforms of both points, the wavelet scalogram for the point from the bottom-right of the wall

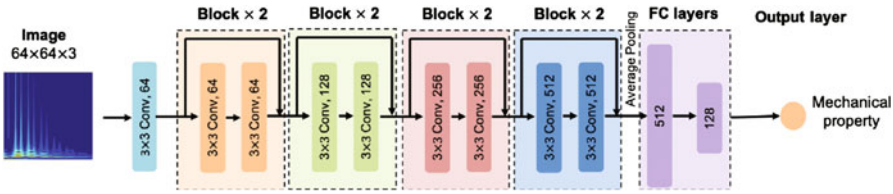

Fig. 7.15 The architecture of the proposed CNN architecture. The first convolution layer has 64 3×3 filters. For each block in the center of the network, the filter size and filter number are shown in the figure. For example, the first residual block has 64 3 × 3 filters, and the last residual block has 512 3 × 3 filters. After eight residual blocks, a global average pooling with four strides was used to reduce the dimension. Then 2 fully-connected (FC) layers with 512 neurons for the first layer and 128 neurons for the second layer were used to make the final prediction

shows comparatively more high frequencies. Both of these plots have a common fundamental frequency of approximately 0.1 Hz. This frequency is related to the scan speed of the AM process.

Step 5: Deep learning for regression and classification

ResNet18 [3], an 18-layer CNN, is used as the base structure as shown in Fig. 7.15. In the first convolution layer, the filter size is 3 × 3 and the stride is 1 and the padding is 1. These parameters help the network maintain most of the information from the inputs. Eight residual blocks are used as the main structure of our network. Each residual block has a residual connection (or shortcut connection). This technique can improve the feature extraction capacity and avoid vanishing gradient problem at the same time. After eight residual blocks and the global average pooling layer, two fully-connected (FC) layers are used to fit the output label. ReLU activation functions are only used in the first two FC layers. The network is used using the Adam optimizer [4] with 1×10^{-3} weight decay. Mean square error (MSE) is used as the loss function. There are several hyper-parameters including learning rate (1×10^{-3}), the number of epoch (50), batch size (8). Besides, fivefold cross-validation is used to choose the optimal network. Thus, the total dataset is split into three parts: training set (64%), validation set (16%), and test set (20%).

Step 6: System and design

Figure 7.16 shows the predicted UTS maps for three process conditions. The three UTS maps on the left denote the original average outputs of the CNN models and the three maps on the right are the associated locally averaged results for clearly demonstrating the spatial variation of the UTS distribution. The two maps in the first-row are associated with the AM process without intentional dwell process and melt pool control. The two maps at the second-row are associated with the AM process with 5 s dwell time between layers but without melt pool control. The third-row maps are associated with the AM process without dwell time but with melt pool control (see the reference[1] for more details). The CNN computed UTS (in black) and experimental values (in red) at positions of interest are marked in Fig. 7.16 as well. The proposed data-driven approach can predict the UTS very

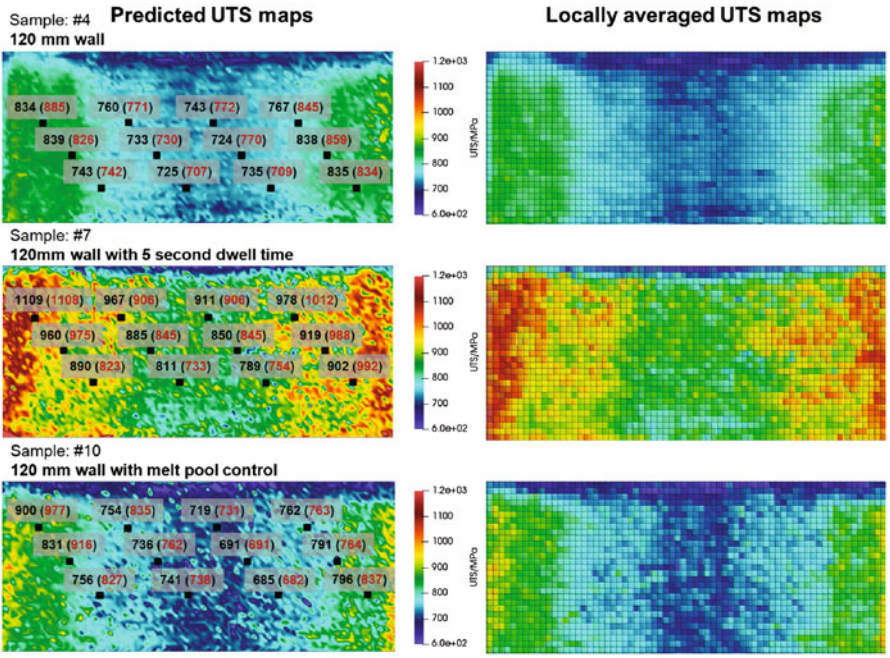

Fig. 7.16 Predicted UTS maps for three process conditions: 120 mm wall without any dwell time and melt pool control, 120 mm wall with 5 s dwell time and 120 mm wall with melt pool control. The CNN outputs (in black) and experimental values (in red) are marked as well

well as compared with the experimental measurements. Leveraging the proposed data-driven approach with high-fidelity simulations, spatial and temporal mechanical properties could be predicted. This methodology provides a mechanistic data-driven framework as a digital twin of physical AM process. It will significantly accelerate AM process optimization and printable material discovery by avoiding an Edisonian trial and error approach.

7.5 Spine Growth Prediction (Type 2 Advanced)

Irregular spine growth can lead to scoliosis, a condition where a side-to-side curvature of the spine occurs. The cause of most scoliosis is unknown, but it is observed in approximately 3% of adolescents [5]. This is known as adolescent idiopathic scoliosis (AIS).

A pure physics-based analysis of the spine and this condition are currently not possible because of the complicated nature of the spinal materials and the slow progress of the condition. Historically, the progression of scoliosis has been assessed through a series of "snapshots" taken through X-rays. These are useful for charting

Fig. 7.17 Front and side X-rays for observing scoliosis

changes, but do not provide much detail related to the interaction between the vertebrae. This is a classic Type 2 mechanistic data science problem in which both the data and the fundamental physics are needed to analyze the condition.

Step 1: Multimodal data generation and collection

X-ray imaging is a common way of assessing the progress of AIS. X-rays are taken from the front and the side in intervals specified by the doctor. The X-rays project a 2D shadow image of the 3D object being evaluated (Fig. 7.17). These two image projections establish the position of the spine at a given instant in time and allow for further measurements of the progression of AIS.

In addition to the X-rays, 3D CT scans can be performed to provide much more detail, but also results in a much higher dose of radiation.

Steps 2, 3 and 4: Extraction of mechanistic features, knowledge-driven dimension reduction, reduced-order surrogate models. As with the piano wave form analysis, the lines between these three steps are somewhat blurry and they will be described together

As described in Chap. 4, the 2D projection images can be used to compute the location of the vertebra of interest in three dimensions. This begins with the use of the snake method to create landmarks outlining each vertebra [6]. The landmark positions in the two projections are used in conjunction with the 3D reconstruction described in Chap. 4 to establish a three-dimensional bounding box for each vertebra. The 3D reference ATLAS model is deformed using the generated landmarks of the X-ray images to establish a 3D model of the vertebrata corresponding to the X-ray images (Fig. 7.18). The data for the 3D reduced order model were then used to refine the dimension of a 3D detailed ATLAS model of the vertebrae (Fig. 7.19).

Step 5: Deep learning for regression and classification

The updated 3D model of the vertebrae as shown in Fig. 7.19 was used to create a finite element model of the spine. These models were used to compute the contact pressure at key landmark locations of the surface of the vertebrae. The predicted contact pressures from the finite element model were combined with the clinical measurements of spine growth in a neural network. This allowed for a more patient-specific prediction of vertebrae growth (Fig. 7.19).

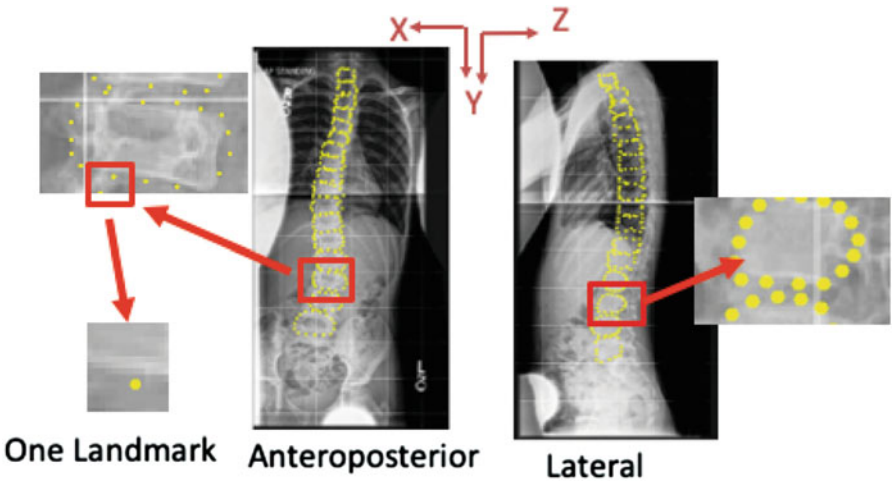

Fig. 7.18 Sixteen landmarks (yellow dots) are located around each vertebra in each projected image

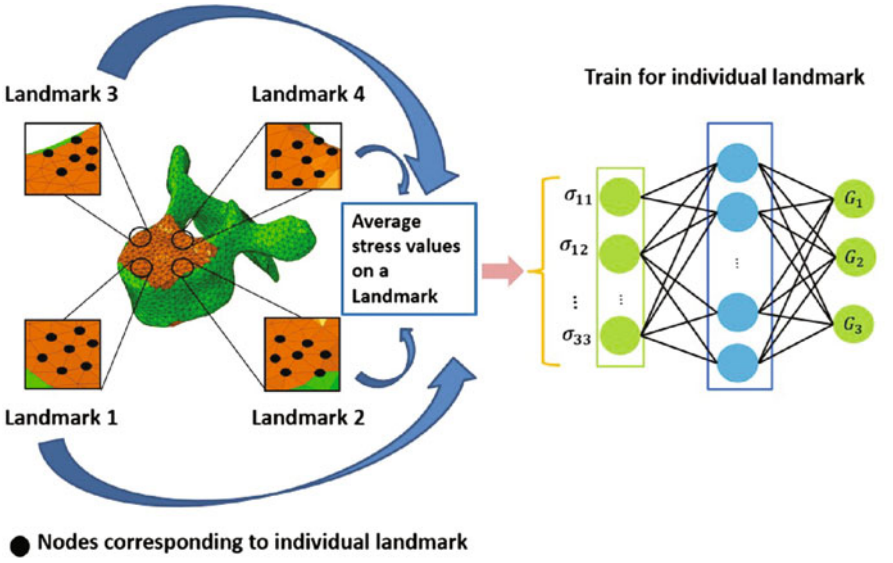

Fig. 7.19 Finite element stress computation at landmarks on the vertebrae used as input to a neural network to predict growth

Step 6: System and design

The data collected and generated based on the X-ray imaging are coupled with predicted stress data at landmark locations to generate a neural network that allowed for prognosis patient-specific spinal growth and deformity (Fig. 7.20). The framework was tested on a single patient as shown in Fig. 7.20. The 3D

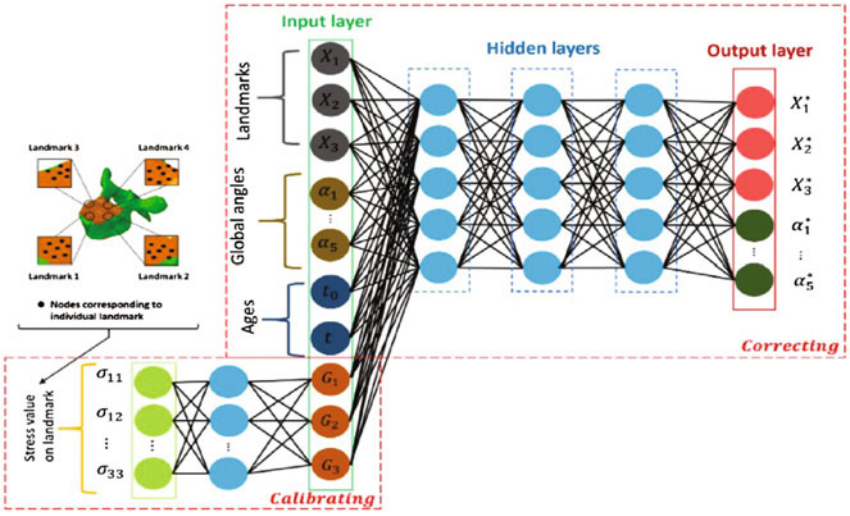

Fig. 7.20 Neural network framework for predicting spinal curvature using mechanistic data science methodology

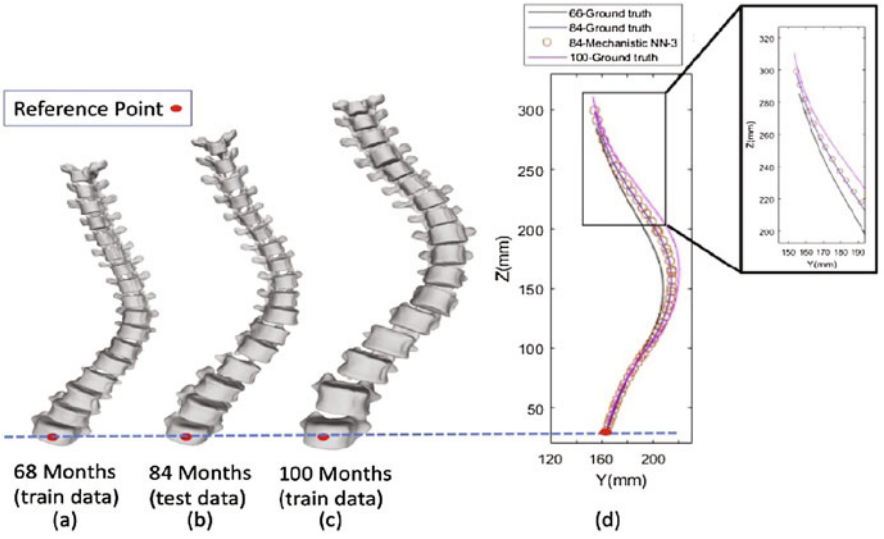

Fig. 7.21 Reconstructed 3D geometry for a patient at (**a**) 68 months, (**b**) 84 months, (**c**) 100 months. (Source: Tajdari et al. 2021—see Footnote 2)

reconstructions of the training data (68 and 100 months) are shown in Fig. 7.20 along with 3D reconstruction of the prediction results inside the ranges of the training data using the Mechanistic NN, which indicates a good agreement with the experimental data (Fig. 7.21).

7.6 Design of Polymer Matrix Composite Materials (Type 3 Advanced)

Materials engineers are constantly challenged to decide which material suits a specific purpose. This requires an in-depth knowledge of material properties for a wide variety of materials available. The material choices can be narrowed down by defining some design and application parameters, such as density, moisture tolerance, or high strength. For instance, choosing a polymer composite for an airplane wing provides better strength to weight performance than a metal and metallic alloy materials. In Chap. 1, polymer composites are briefly discussed and the method to make a composite material is demonstrated. Given a desired composite property, the way to choose a specific combination of the constituent phases (matrix and fibers) will be shown through a system and design problem. First, a system and design problem will be discussed, and then the steps of mechanistic data science that can be carried out to solve the problem will be shown. Suppose an engineer requires certain composite materials properties such as elastic modulus, resilience, toughness, and yield strength. The materials engineer is asked to provide the matrix and the fiber type with a specific composition to achieve the desired properties. This poses several challenges since a materials engineer must consider many different parameters, such as the material combinations, the constituent fractions of each material, and the temperatures of the material. This process can be very time-consuming and costly, but mechanistic data science can help streamline the process by learning the relationship from a few material combinations in order to derive a solution. Using mechanistic data science, the problem boils down to how to learn the hidden relationship of materials system, microstructure, and their response from limited available data [7]. That knowledge can then be used to find an appropriate materials system (Fig. 7.21).

Step 1: Multimodal data generation and collection

The first step is to collect or generate data that is relevant to the problem. The properties of interest can be found from the stress-strain data for a composite materials system. Composite microstructures with fiber volume fractions from 1 to 50% have been prepared to perform numerical tensile testing to predict the stress-strain data. Self-consistent clustering analysis has been used to accelerate the tensile test data generation process [8]. Some data can also be generated by other computer simulations such as finite element analysis. Stress-strain data can further be collected through physical experiments, although sample preparation can be costly. Four different fiber materials were tested in combination with three different matrix materials. The four fibers are carbon, glass, rayon and Kevlar. The three matrix materials are epoxy, poly (methyl methacrylate) (PMMA) and polyethylene terephthalate (PET). Numerical tensile testing is carried out by applying a tensile load along the fiber direction up to a strain value of 0.04 for three different temperatures. The data generation process and mechanistic feature extraction is demonstrated in Fig. 7.22.

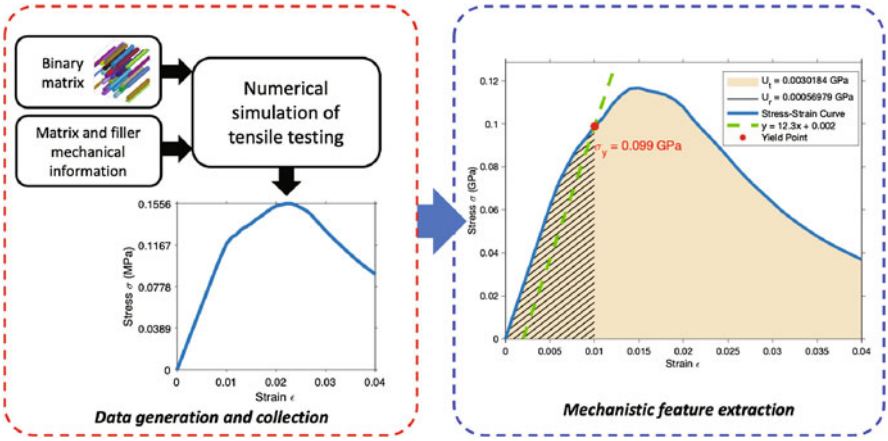

Fig. 7.22 Stress-strain data generated from tensile simulations. Yield strength, elastic modulus, toughness, and resilience have been extracted as mechanistic features from the stress-strain data

Step 2: Mechanistic feature extraction

The stress-strain curves generated from the numerical tensile simulations show several mechanistic features (namely elastic modulus, yield strength, toughness, and resilience) that can be extracted. *Elastic modulus* is defined as the slope of the linear (or elastic) portion of the stress-strain curve. *Yield strength* is the onset of nonlinearity of the stress-strain curve which is sometimes hard to identify on the curve. One popular way to determine when the stress-strain curve becomes nonlinear is to take 0.2% strain offset of the linear portion of the stress-strain curve. *Toughness* is a measure of the material's ability to absorb energy, and it can be calculated from the area under the stress-strain curve. The *resilience* is the elastic energy stored during the loading, and it can be obtained by finding the area under the elastic portion of the stress-strain curve. All these mechanistic features of the stress-strain curve depend on the microstructure features such as volume fractions of the fiber and the fiber and matrix materials system.

Steps 3 and 4: Knowledge-driven dimension reduction and reduced order surrogate models

Understanding a relation between the microstructure features, materials property features, and stress-strain curve features are essential to solve the problem. However, considering the materials properties, the problem is very high dimensional as there are a total of 17 features (see Fig. 7.23). To reduce the number of material features, they are divided into three categories: Microstructure Descriptor (MSD), Mechanical Property Descriptor (MPD), and Physical Property Descriptor (PPD). These categories were determined to hold the most related features, which would be useful in dimension reduction. MSD includes the volume fraction (ϕ), matrix density (ρ_m), and filler density (ρ_f), which describe

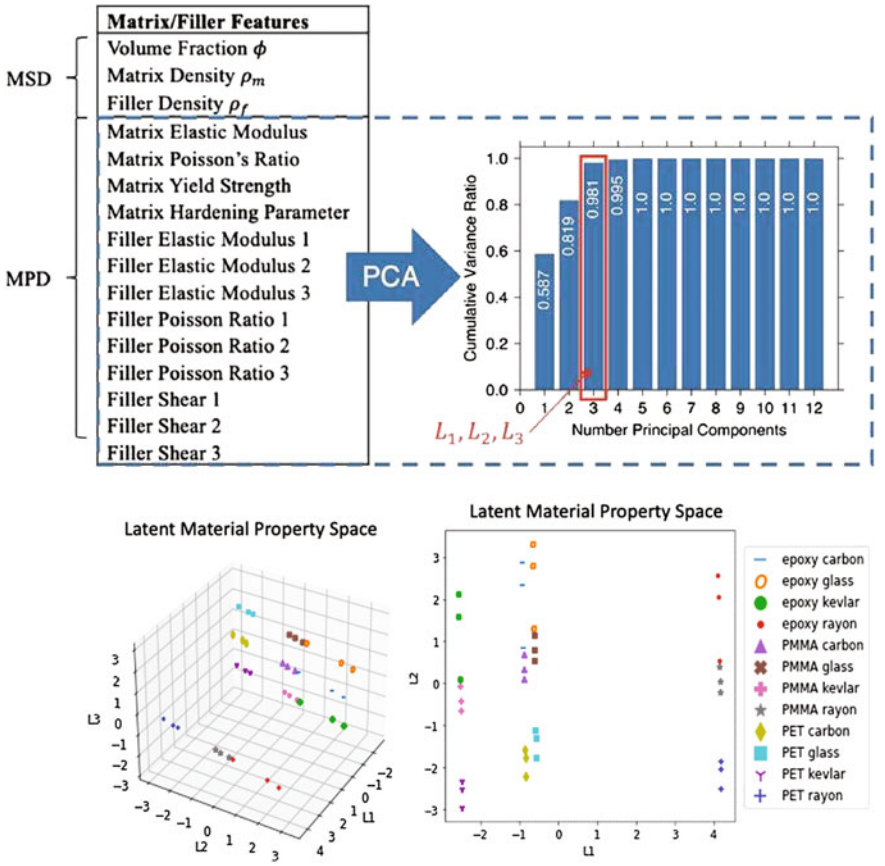

Fig. 7.23 Very high dimensional materials feature that can be compressed using PCA. Three PCA components can represents 98.1% of the MPD data. The latent material property space is the space created by the three main principal components of the MPDs. Bottom left figure shows the 3D space created with L_1, L_2, and L_3, while bottom right shows the 2D space created only with L_1 and L_2

the densities and ratio makeup of the composites. MPD includes the rest of the filler/matrix features, which describe the mechanical properties of the matrix and filler materials. PPD only consists of temperature, as it describes the physical surroundings of the composite.

The features in MSD could all be related to a fourth variable, composite density (ρ_c), using the "rule of mixtures"

$$\rho_c = \rho_f \phi + \rho_m (1 - \phi),$$

which says that the composite density is equal to the weighted sum of the filler and matrix densities. Therefore, all three MSD features can be reduced to ρ_c.

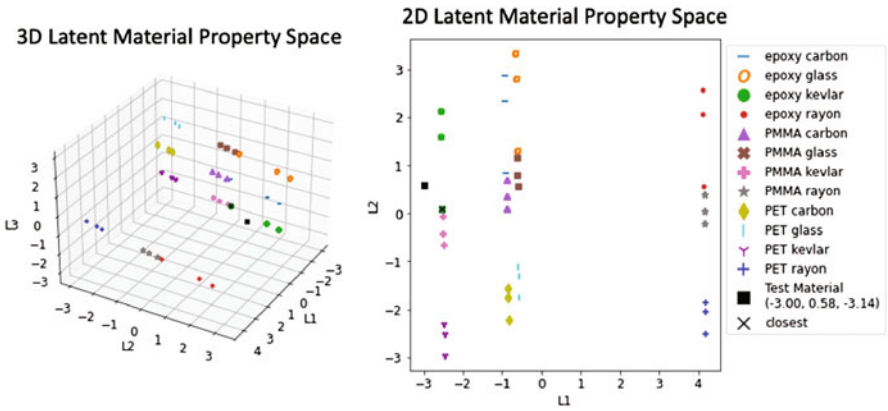

Fig. 7.24 The principal component values of the material with maximized mechanical features and a low density on the LMPS

The MPD features are reduced using Principal Component Analysis (PCA). PCA is a process that transforms a dataset's space so that each dimension, called principal components, is linearly independent. It was determined that 98.1% of the data could be described by three principal components, which will be referred to as L_1, L_2, and L_3. All of the data was graphed in this 3D space created by the principal components (see Fig. 7.24), which will be called the "Latent Material Property Space" (LMPS). When looking at the 2D LMPS in Fig. 7.24, the clustering of the LMPS suggests that L_1 represents the filler material, while L_2 represents the matrix material. There are three values of L_2 per cluster because there is data for three different temperatures, and the matrix properties are temperature dependent. This interesting clustering pattern implies that if given a set of unknown matrix and filler properties, the principal component values could be plotted on the LMPS to find similar materials from the known database.

Step 5: Deep learning for regression

A feed-forward neural network (FFNN) is used, with the desired mechanical features and density as inputs, and the principal component values of the matrix and filler properties as outputs. The final FFNN model is comprised of three hidden layers, with 50 neurons per layer. The hidden activation functions are sigmoid, while the output activation function is linear. The Adam optimizer is used, with a learning rate of 0.001. The model was trained in minibatch sizes of 32 for 4233 epochs. The resulting model had a MSE of 0.271 (validation MSE of 0. 286) and an R^2 score of 0.9601.

Step 6: Design of new composite

The principal component outputs of the FFNN can be compared to the LMPS, which will show the closest material system. These outputs can also be turned back into their matrix and filler properties with reverse PCA. The flowchart of the process is shown in Fig. 7.25.

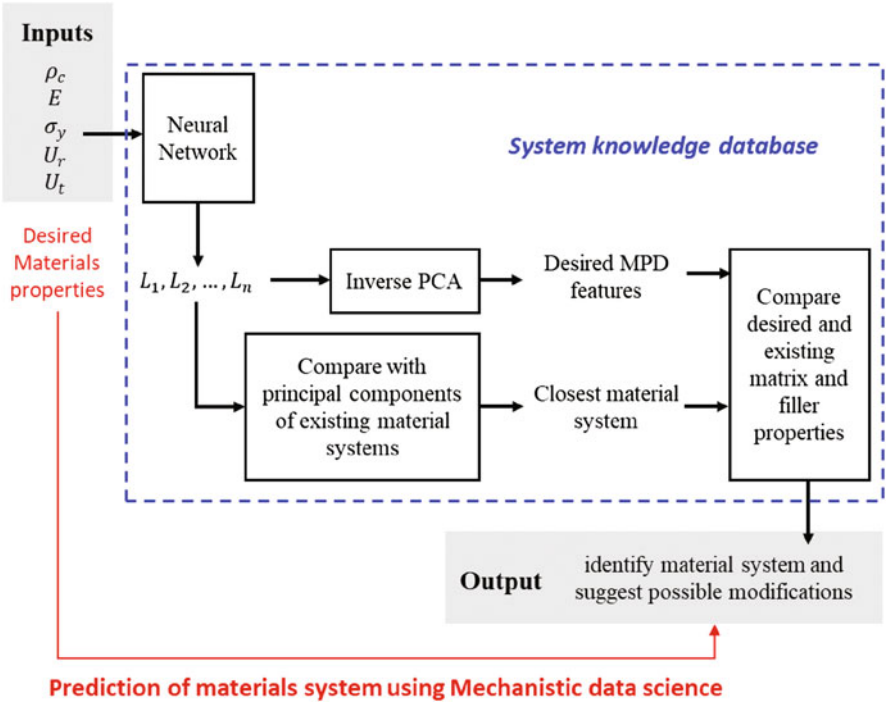

Fig. 7.25 Flowchart of how to predict a specific material system using the proposed model

For one test, the desired mechanical features chosen are the maximum values of elastic modulus, yield strength, resilience, and toughness in the dataset, while choosing a low density of 1300 kg/m^3. By putting these values through the FFNN and comparing the outputs to the LMPS, this material is predicted to have matrix and filler materials closest to epoxy Kevlar at 295 K. A comparison of the desired material properties and epoxy Kevlar properties at 295 K and 0.055 volume fraction is shown in Table 7.3. Many of the matrix and filler properties are quite similar, but the matrix elastic modulus and matrix yield strength of the desired material is significantly higher than that of the epoxy Kevlar. To increase these values, many studies have shown that adding silica nanoparticles to epoxy matrices both increased the elastic modulus and yield strength of the matrix [9].

Table 7.3 Desired material properties vs. epoxy Kevlar properties at 295 K and 0.055 volume fraction

Features	Desired material	Epoxy Kevlar 295 K
Volume fraction	0.055556	0.055
Matrix elastic modulus (MPa)	6238.186	4408
Matrix Poisson's ratio	0.385362	0.4
Matrix yield strength (MPa)	89.313338	69.79866
Matrix hardening parameter	0.297894	0.446773
Filler elastic modulus 1 (MPa)	166,939.45	150,000
Filler elastic modulus 2 (MPa)	3228.879	4200
Filler elastic modulus 3 (MPa)	3228.879	4200
Filler Poisson 1	0.362125	0.35
Filler Poisson 2	0.362125	0.35
Filler Poisson 3	0.354568	0.35
Filler shear 1 (MPa)	2646.434	2900
Filler shear 2 (MPa)	2646.434	2900
Filler shear 3 (MPa)	1051.689	1500
Composite density (kg/m^3)	1300	1299.9
Composite elastic modulus (MPa)	24.013	4.844093
Composite yield strength (MPa)	0.164331	0.074844
Composite resilience (MPa)	0.001400	0.000707
Composite toughness (MPa)	0.005218	0.002826

7.7 Indentation Analysis for Materials Property Prediction (Type 2 Advanced)

Indentation is a useful technique to extract mechanical properties of materials. This is a low-cost semi or nondestructive testing procedure that is less time-consuming than tensile testing and capable of providing important materials properties such as hardness and elastic modulus. Knowledge of mechanical properties of materials is essential for engineers to design any real-life part or products. In indentation experiment, an indenter of known shape (e.g., spherical, conical, etc.), size, and materials is penetrated through the workpiece materials and the load-displacement data is recorded for both loading and unloading of the indenter through the testing workpiece. The materials properties are extracted by analyzing the load-displacement curve (also known as P-h curve). Important materials physics such as plasticity, yielding occurs during this loading, unloading step and P-h curves carries the signature of the materials localized properties. Several other mechanisms also get activated during the indentation test. For metal and alloys, dislocations are generated and propagated during the loading step which deforms the materials permanently in a form of line defects. Dislocation mechanism can provide many important aspects of the materials failure eventually.

Indentation test provides the hardness data for a material at the location the materials is tested. Hardness is thus a localized property that varies from point to point of the materials. Hardness can be directly related to other mechanical properties of the materials such as yield strength, elastic properties, and hardening parameters. Predicting materials properties from the localized hardness data is referred as the inverse problem of the indentation. This has a direct application on additively manufactured materials. With the process variability of the additive manufacturing process the local microstructures alter significantly that gives a property variation of the AM built parts. To ensure the part integrity it is very important to evaluate the localized mechanical properties and relate them to the processing conditions. However, performing tensile experiments in microstructure level samples are hard and it takes a long time. Instead, nano or instrumented indentation is an easier alternative to evaluate localized mechanical properties of the materials. Using mechanistic data science, inverse problem of the indentation can be solved and a relation between the localized hardness and yield strength can be established for an AM build Ti-64 alloy parts.

Step 1: Multimodal data generation and collection

Nanoindentation data has been collected for additively manufactured Ti64 alloy from the literature [1]. The summary of the available data set is given in Table 2.1 in Chap. 2. The data set consists of indentation data for AM built part for different processing conditions. For this example, S3067 processing conditions having 144 indentation tests data has been selected as the experimental data set. The load-displacement data from the experiment can be considered high-fidelity testing data. However, the localized mechanical properties such as yield strength is not evaluated locally. For this purpose, 70 numerical indentation simulations have been carried out considering different materials property as input (solving the forward problem) (See animation in Chap. 2). From both experiment and numerical simulation load-displacement data is obtained and stored in a database. Our objective is to use this multimodal data to establish the relation of the experimental low resolution materials property data with indentation testing data (hardness) first and then transfer it for simulation data using transfer learning to find the materials property (yield strength and hardening parameters) from an inverse problem.

Step 2: Mechanistic features extraction

Several mechanistic features needed to be extracted from the collected load-displacement data (see Fig. 7.26). For instance, load-displacement can be used to find the peak load and the indentation area which is necessary to calculate the hardness and other mechanical properties. From the loading portion of the curve, a curvature (C) is identified. From the unloading the slope (S) of the unloading curve at the maximum load point is identified and a contact area is calculated based on the residual depth, h_c. The area under the loading curve is the total work done, while the area under the unloading curve is due to the elastic work. The ratio of plastic work (difference between total to elastic work) to total work ($\frac{W_p}{W_t}$) is

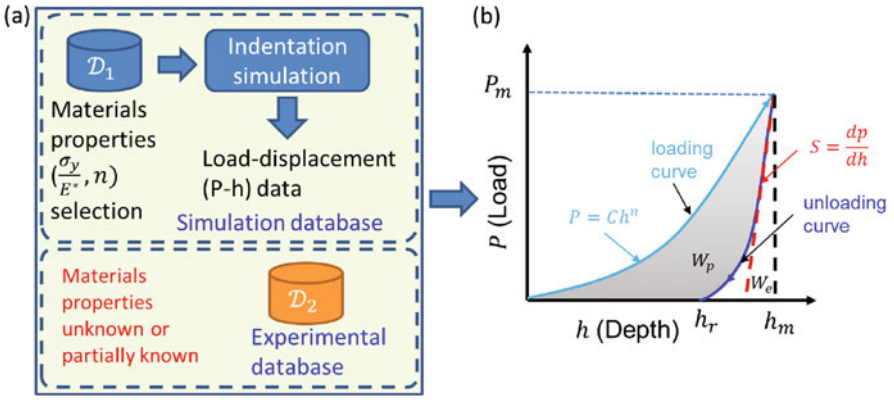

Fig. 7.26 (**a**) Multimodal data generation and collection for indentation test, (**b**) typical load-displacement data from an indentation test with different mechanistic features

considered as another important features. From the load displacement data, the hardness can be calculated using the following formula,

$$H = P_m / A_c$$

where P_m is the maximum load during the indentation process and the contact area for a Berkovich indenter can be obtained by

$$A_c = f(h_c) = 24.56 h_c^2.$$

The critical depth—h_c is measured from the slope of the unloading curve using the formula,

$$h_c = h - \frac{\varepsilon P_m}{S}$$

Another important feature is indentation reduced modulus that can be evaluated from the following equation,

$$E^* = \frac{s}{2\beta}\sqrt{\frac{\pi}{24.56}}$$

where β depends on the indenter shape and for Berkovich indenter it has a value of 1.034.

Finally, the elastic modulus of the workpiece and the indentation reduced modulus can be related by

Table 7.4 Mechanistic features and their physical units

Mechanistic features	SI unit
Curvature, C	Pa (N/m^2)
Slope of unloading curve, S	N/m
Maximum load, P_m	N
Maximum depth, h_m	m
Plastic to total work ratio, $\frac{W_p}{W_t}$	–
Hardness, H	Pa
Reduced modulus, E^*	Pa
Yield strength, σ_y	Pa
Hardening parameter, n	–

$$\frac{1}{E^*} = \frac{1-\nu_s^2}{E_s} + \frac{1-\nu_i^2}{E_i}$$

where subscript "s" denotes the workpiece and "i" denotes the indenter.

Therefore, from the load displacement data, following mechanistic features can be identified: C, P_m, h_m, h_c, A_c, S, H, E^*

Steps 3 and 4: Knowledge-driven dimension reduction and reduced order surrogate models

Instead of working with all the features extracted, a dimensional analysis can be performed on the features from the load-displacement curve. The process of the nondimensional has been explained in Chap. 5 and same concept can be applied on the nondimensionalization of the indentation features. Table 7.4 describes different mechanistic features and their units. For further study of the nondimensionalization, interested readers are referred to Cheng and Cheng [10].

Using these important features, six important nondimensional groups can be identified. Among them four are obtained from the indentation test and two are materials properties. The nondimensional groups can be written in a functional form of the materials properties as follows:

$$\frac{C}{E^*} = \Pi_1\left(\frac{\sigma_y}{E^*}, n\right)$$

$$\frac{S}{E^* h_m} = \Pi_2\left(\frac{\sigma_y}{E^*}, n\right)$$

$$\frac{W_p}{W_t} = \Pi_3\left(\frac{\sigma_y}{E^*}, n\right)$$

$$\frac{H}{E^*} = \Pi_4\left(\frac{\sigma_y}{E^*}, n\right)$$

These nondimensional groups are highly correlated with each other and can be used for scaling analysis. These relations of the indentation features with the

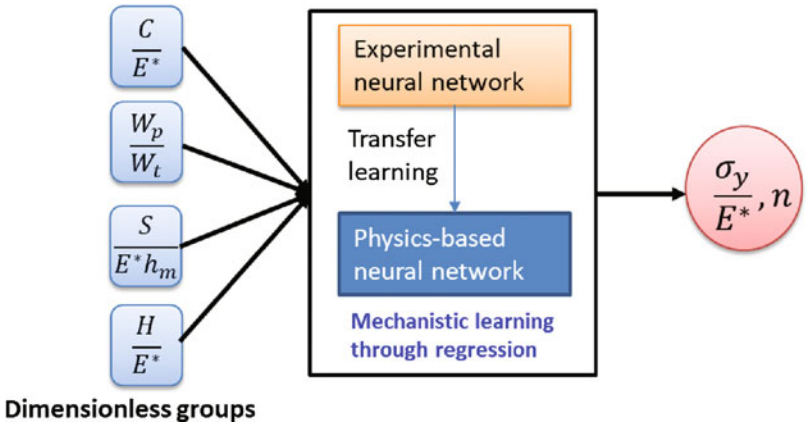

Fig. 7.27 Mechanistic data science approach to predict localized materials property from indentation analysis

materials properties can be put in a functional form from the above equations as follows.

$$\left(\frac{\sigma_y}{E^*}, n\right) = f\left(\frac{C}{E^*}, \frac{S}{E^* h_m}, \frac{W_p}{W_t}, \frac{H}{E^*}\right)$$

This relationship further reduces the dimension of the problem. This relationship can be found using a neural network based surrogate model which is the next step of this problem.

Step 5: Deep learning for regression

As shown in Fig. 7.27, a relationship among the experimentally obtained data and simulation data needed to be established since experimental local mechanical properties such as yield strength are not known. However, the properties are used to generate data from the simulations. A transfer learning approach is applied to build a better model to have a better prediction of the hardness from the input materials properties. First, an experimental neural network having the nondimensional input variables shown in Fig. 7.28 is trained for the output of materials properties. This neural network has two hidden layers with 50 neurons each and "ReLU" type activation function. Training, testing, and validation have been set as 70%, 20% and 10% for the dataset. The R^2 value obtained for this neural network is 0.70. Later these pretrained layers are used for the data generated from the simulation to train a separate neural network. This physics-based neural network has two additional hidden layers having 20 neurons each. The R^2 value increased up to 0.74 for the physics-based NN. Finally, this neural network can take any indentation test features as input (in dimensionless form) and predict the localized properties.

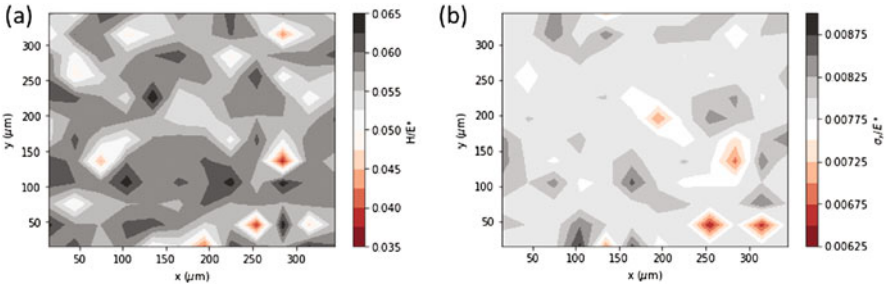

Fig. 7.28 (**a**) Nondimensional hardness distribution over the AM sample surface, (**b**) MDS predicted localized yield strength mapping

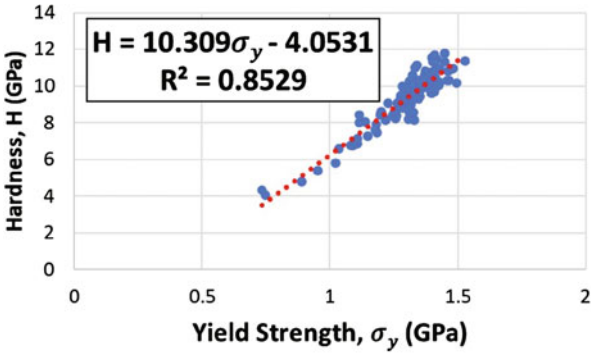

Fig. 7.29 Relation between hardness and yield strength for Ti64 alloys printed following S3090 printing conditions

Step 6: System and design for new materials system

Using the previous steps of mechanistic data science, a knowledge base for the indentation system is set up and then a new system can be tested. To illustrate this, a new indentation test data has been selected from [1] for S3090 processing conditions. The sample size is 360 μm by 360 μm and an indenter is indented every 30 μm apart. The nondimensional hardness distribution for the sample is shown in Fig. 7.28. In Fig. 7.28b the MDS predicted localized yield strength has been shown. The hardness and the yield strength are highly correlated, and a further analysis provides a mechanistic relation for the hardness and yield strength for such specific materials system (see Fig. 7.29).

7.8 Early Warning of Rainfall Induced Landslides (Type 3 Advanced)

A landslide occurs when the soil and rocks on a hillside give way and a large section of the hillside suddenly moves down the hill. There are multiple factors that affect the likelihood of having a landslide, including soil type, rainfall, and slope

Fig. 7.30 Landslide along California's Highway 1. (Photo credit San Luis Obispo Tribune)

inclination. As shown in Fig. 7.30, landslides can be very damaging to property and threaten the lives of people in their vicinity. The cost of landslides damage to public and private property in the United States is estimated to exceed $1 billion per year [11].

Landslide Early Warning Systems (LEWS) attempt to mitigate the threat of landslides by monitoring key variables and providing a timely warning to a population. One key parameter is soil moisture due to rain and storms, which inspires the construction of precipitation intensity-duration thresholds for shallow, rainfall-induced landslides. However, other factors come into play as well, which explains why thresholds developed from limited historical databases of rainwater infiltration have large variability in landslide occurrence times. This historical data usually does not show how factors such as topography, soil properties, and soil initial conditions affect reported landslide times.

Mechanistic data science can be used to analyze the interplay of numerous key parameters and conditions for landslide prediction. This example shows how rainfall-induced landslide predictions can be improved by combining historical data with data generated from water infiltration simulations. The MDS steps are outlined below:

Step 1: Multimodal data generation and collection is the important first step to ensure that sufficient data is available for analysis. Properties of interest include failure time, rainfall intensity, soil cohesion, soil porosity, soil density, initial moisture conditions, and slope angle. Failure time is the time it takes for a landslide to occur. The rainfall intensity is the amount of water incident on a soil per unit time (mm/h). Soil cohesion is the measure of the force that holds together soil particles. Soil porosity is the percentage of air or spaces between particles of soil in a given

sample. Soil density is the dry weight of the soil divided by its volume. The initial moisture conditions are represented by the difference in weight of the soil dry and weight of the soil when moist. The slope angle is the angle measured from a horizontal plane to a point on the land. Note that many historical landslide databases do not include all these parameters. For situations where all the data are not accessible, computer simulations can be used to generate some data. This includes simulating water infiltration events with different parameters to compute the time for the slope to become unstable. Data was collected through a physically-based simulation software [12, 13]. The simulation software used comes with a physical evaluation of a factor of safety threshold [14] to determine when the landslide occurs based on the moisture content throughout the soil column. All of the properties of interest above are obtained through this software. Figure 7.31 shows how the simulation software generates data through time.

Data were extracted from the database for these soil parameters. Figure 7.32 shows plots of failure time versus rainfall intensity for multiple slope angles. It can be seen that the time for a landslide to occur decreases as the rainfall intensity

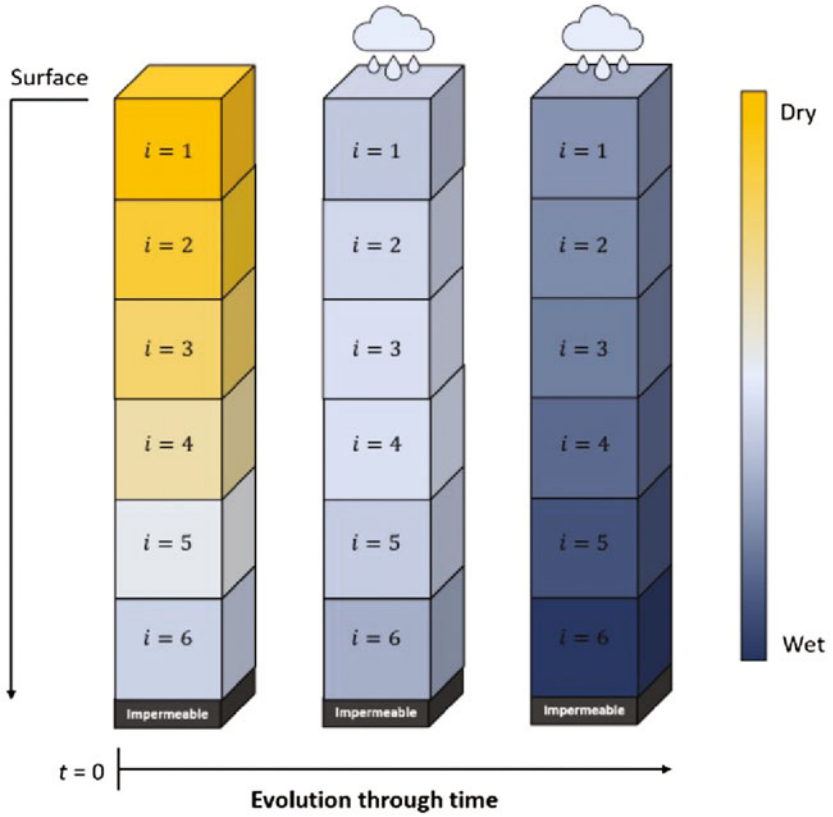

Fig. 7.31 A diagram illustrating a soil column with a simulated rainfall event

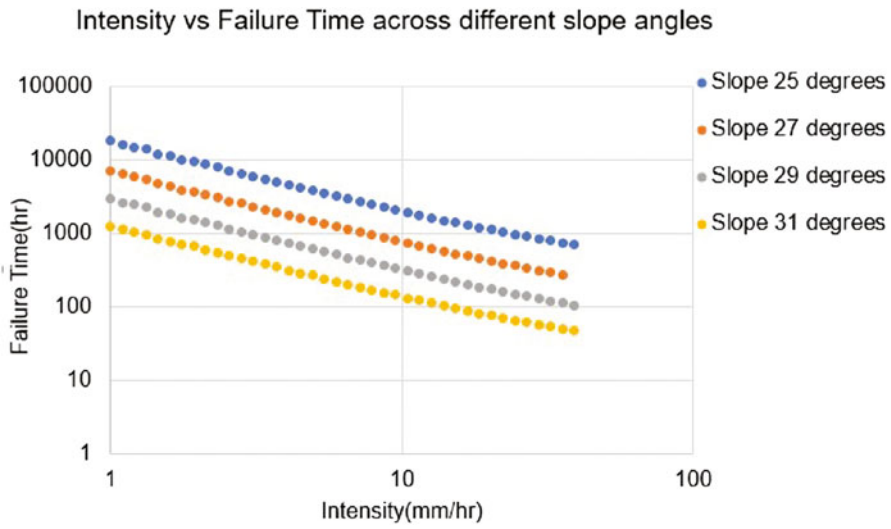

Fig. 7.32 Failure time vs. Intensity across several slope angles

increases. It can also be seen that steeper slope angles will results in faster landslides at all rainfall intensities the soil cohesion is quantified through the internal friction angle, as suggested by the literature. Because this angle is 37.24°, any data near it or above this level will have a skewed response in a different domain from the rest of the data. As a result, the choices of slope angle will be in the range of 25–35°. The log of both intensity and failure time are then taken on all datapoints because past research shows a logarithmic relationship between these variables.

Step 2: Extraction of mechanistic features determines important characteristics from the data collected and generated. From these features, rainfall intensity and slope angle were chosen as inputs while failure time was chosen as output. These two inputs were picked because they are more easily measured compared to specific soil parameters. Table 7.5 below shows each feature obtained from the simulation software as well as their units and range. The bottom four parameters are all defined as a constant while rainfall intensity and slope angle differ to produce a landslide failure time.

Steps 3, 4: knowledge-driven dimension reduction and reduced order surrogate models

Each soil parameter was standardized to a constant. This was done to reduce the amount of input variables while retaining the relationship between inputs and outputs. All the soil properties and external conditions (rainfall intensity and slopes) mentioned in the Table 7.4 above are important parameters for failure time prediction. Rainfall intensity and slope angle are reported in the literature as the most important variables associated with landslide triggering since they can

Table 7.5 Features extracted from the simulation software

Parameters	Units	Range
Failure time	h	1–20,000
Rainfall intensity	mm/h	1–40
Slope angle	°	25–35
Soil cohesion	°	37.24
Porosity	%	71.5
Soil density	g/cm^3	8.7
Initial soil moisture content	kPa	3.57

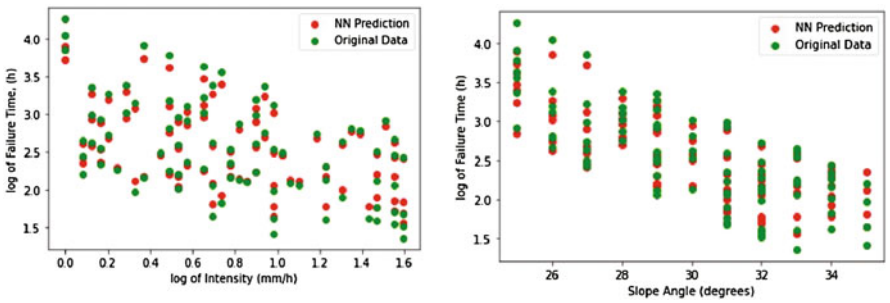

Fig. 7.33 Intensity and slope angle vs. failure time in neural network

vary a lot: every rainfall can have a different intensity, and a hill can have different slope angles at every point. Therefore, rainfall intensity and slope angles are varied for data generation and database preparation, which reduces the problem dimension significantly.

Step 5: Deep learning for regression

A feed-forward neural network was constructed to relate the rainfall intensity and slope angle (inputs) to the landslide failure time (output). After randomizing the dataset, *k*-fold cross-validation was conducted. To accomplish this, dataset was divided into 5 groups ($k = 5$). Each group was subject to testing and the other four groups were used to fit the model.

The mean absolute error ranges from 0.04 to 0.05 and the coefficient of determination is $r^2 = 0.96$. Figure 7.33 shows the original data compared to the neural network predictions. It can be seen that the neural network accurately predicts the landslide failure time for both the rainfall intensity and slope angle.

Step 6: System and design for intensity-duration thresholds

The neural network predicts the failure time given input intensity and slope angle. This analysis builds on the premise of having relatively static soil parameters. The slope angle of a given area is available to be measured. The trained neural network can then be used to compute the rainfall intensity-duration thresholds that indicate landslide risk. This analysis can be applied to the coastal and mountainous areas of California, which are some of the areas most susceptible

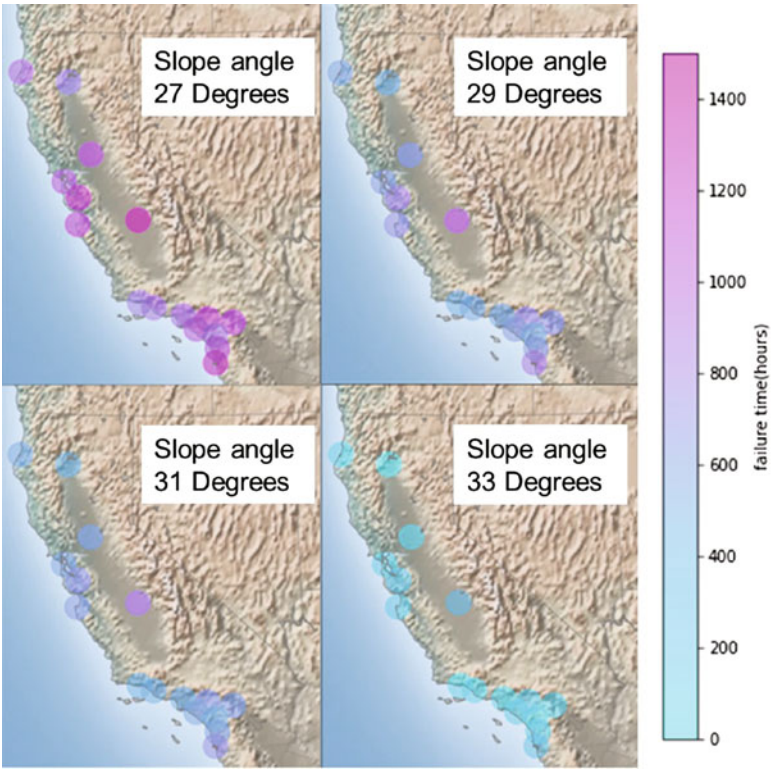

Fig. 7.34 Landslide failure times of California cities considering different slope angles and rainfall intensities

to landslides in the United States. Figure 7.34 identifies several locations throughout the state for analyzing landslide risk. Rainfall intensity at these California locations can be used to predict possible landslides for areas with different slope angles.

7.9 Potential Projects Using MDS

7.9.1 Next Generation Tire Materials Design

One of the fundamental questions that tire industry faces is on the durability of the tire. The unpredictable weather conditions and road surface conditions that each tire faces every day have a significant impact on its durability. One of the key materials property metrics that can be related to the tire material performance is the tan(δ). For tire material it is desirable that it has high tan(δ) for low temperature (provide better ice and wet grip) and low tan(δ) at high temperature (provide better rolling friction).

It is noteworthy to mention that approximately 5–15% of the fuel consumed by a typical car is used to overcome the rolling friction of the tire on the road. Therefore, controlling the rolling friction of tires is a feasible way to save **energy** (by reducing fuel consumption) **and environment** (by reducing carbon emission). We can also ensure the **safety operation** of tire providing sufficient ice or wet grip.

The key performance metric, $\tan(\delta)$, is a function of the matrix materials, microstructure, and the operating conditions such as temperature and frequency. It is well known that adding filler improves the tire materials performance. But what fillers and their distribution to achieve optimized properties and performance is still an important research question. The design space combining different rubber matrices and fillers, microstructure and operating conditions can be so enormous that experimental or simulation technique is not feasible at all. The mechanistic data science approach can provide an effective solution to explore the design space leveraging the data science tools through revealing the mechanisms and construction of necessary accurate and efficient reduce order surrogate model. This approach will enable the industrial practitioner to perform rapid design iteration and expedite the decision-making process. The six modules of the mechanistic data science approach to tackle the problem.

Multimodal data generation and collection: Experimental data such as materials composition, microstructure images along with mechanical testing such as tension, shear, DMA, friction tester, etc. can be collected for different matrix and filler combinations. Multiscale simulation data can be generated through the numerical simulation. Using transfer learning, we can establish a robust numerical model to generate data which is very accurate compared to the experiment. We also need to account for interphase and interface characterization.

Mechanistic features extraction: Important mechanistic features needs to be identified such as tan delta or others. Microstructure features such as volume fraction, filler distribution, and the operating conditions features also need to be analyzed. How these mechanistic features relate to the materials performance will be analyzed by a sensitivity analysis.

Knowledge driven dimension reduction: Based on materials choice, microstructure features and operating condition features we can reduce the dimension of the problem. Potentially, we can use PCA to transfer many less understood correlations among those features in a latent space and understand which shows similar performance and why.

Regression and Classifications: Relating the mechanistic features with the performance metric such as $\tan(\delta)$. Classifications can be used to better understand local materials performance such as crack initiation, damage, etc.

Reduce order surrogate model: Establishing a mechanistic based ROM for rapid performance metric prediction for a given operating condition.

System and design: Use the ROM for materials multiscale performance prediction and apply this for the real tire design.

7.9.2 *Antimicrobial Surface Design*

Since ancient times, people in South Asia use metal and alloy (such as gold, silver, bronze, steel, etc.) plates for their daily meals. They believed that eating food from a metal plate was good for their health, but most of them did not know why. Though the situation has now changed with the polymeric and ceramic tableware, eating off a metal plate is still very common in many South Asian countries. There is an important science hidden behind this usage of metal plates related to food bacteria. It is well known that active metals like gold and silver have antimicrobial properties. Silver has been widely used considering its antimicrobial effects and low cost. Very recently, it has been reported that copper and copper alloys may have similar or even better antimicrobial effects [15]. This can provide a very low-cost solution to prevent bacterial infections by making coatings in daily-use public places (such as buses, lift buttons, etc.).

These metal and alloy surfaces can kill the bacteria when it comes into contact with the surface. The key term is the "ion diffusion" through bacteria membrane, which eventually kill the bacteria. Increasing the ion diffusion rate can accelerate the bacteria killing. Surface designing parameters such as surface patterning and surface roughness can be directly correlated with the ion diffusion rate for this metal and alloy surfaces. However, identifying the engineered surface to minimize the bacteria growth is not straightforward, as it depends on the kinetics of the bacteria growth and the surface bacteria interaction. Using the MDS framework with available data science tools allows the study of the relation between different alloy designs, their ion diffusion rate in relation with the surface patterning. This will reveal the mechanisms in more detail, and eventually lead to engineered surfaces based on the knowledge gained.

The six steps of mechanistic data science are outlined below for this problem.

Multimodal data generation and collection: For this step, data is collected from several information sources, such as ion diffusion rate of different metal and alloys combination, relation of surface roughness to the ion diffusion rate, bacteria surface interaction, and bacteria growth kinetics.

Mechanistic features extraction: The bacteria growth data are typically collected in image form. Images must be analyzed to identify the key mechanistic features of bacteria growth on such surface. Also, surface roughness features and ion diffusion rates can be extracted from experimentally collected data or modeling.

Knowledge driven dimension reduction: Having the large data set of metal and alloy combination with different bacteria, the problem becomes very high dimensional. Different dimension reduction techniques can be employed to identify the important features to be extracted from a huge dataset.

Reduced order surrogate model: Several reduced order model can be developed for the mechanisms such as the growth kinetics of bacteria for different surface parameters, surface roughness and patterning and ion diffusion mechanism for metal alloys.

Regression and Classifications: Relating the mechanistic features (surface roughness, bacteria growth kinetics, etc.) with the performance metric such as ion diffusion rate will provide necessary information to look at different alloy materials combinations and their antimicrobial properties.

System and design: Develop and design an engineered metal or metal alloy surface which reduces the bacteria growth rate.

7.9.3 Fault Detection Using Wavelet-CNN

A very important application of the wavelet-based CNN methodologies is the fault detection during the non-destructive testing (NDT) of metallic part. For example, detection of welding defects with wavelet-CNN [16]. Welding faults include slags, lack of fusion, and cracks. To detect these faults, an eddy current is applied, and the resulting magnetic field fluctuation is recorded. The response of the magnetic field is different for defect-free and flawed parts. These signals are converted to 2D images using wavelet transformation and finally, a CNN is applied to classify the test to identify what kind of fault is present inside the material.

References

1. Xie X, Saha S, Bennett J, Lu Y, Cao J, Liu WK, Gan Z (2021) Mechanistic data-driven prediction of as-built mechanical properties in metal additive manufacturing. npj Comput Mater 7:86
2. Meyer Y (1992) Wavelets and operators. Cambridge University Press, Cambridge. ISBN 0-521-42000-8
3. He K., Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
4. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
5. https://www.mayoclinic.org/diseases-conditions/scoliosis/symptoms-causes/syc-20350716
6. Tajdari M, Pawar A, Li H, Tajdari F, Maqsood A, Cleary E, Saha S, Zhang YJ, Sarwark JF, Liu WK (2021) Image-based modelling for adolescent idiopathic scoliosis: mechanistic machine learning analysis and prediction. Comput Methods Appl Mech Eng 374(113):590
7. Huang H, Mojumder S, Suarez D, Amin AA, Liu WK (2021) Design of reinforced polymer composites using mechanistic data science framework (in preparation)
8. Liu Z, Bessa M, Liu WK (2016) Self-consistent clustering analysis: an efficient multi-scale scheme for inelastic heterogeneous materials. Comput Methods Appl Mater Eng 306:319–341
9. Domun N, Hadavinia H, Zhang T, Sainsbury T, Liaghat GH, Vahid S (2015) Improving the fracture toughness and the strength of epoxy using nanomaterials—a review of the current status. Nanoscale *7*(23):10294–10329
10. Cheng YT, Cheng CM (2004) Scaling, dimensional analysis, and indentation measurements. Mater Sci Eng R Rep 44(4–5):91–149
11. Fleming RW, Taylor FA (1980) Estimating the costs of landslide damage in the United States. U.S. Geological Survey, Circular 832

12. Lizarraga JJ, Buscarnera G (2019) Spatially distributed modeling of rainfall-induced landslides in shallow layered slopes. Landslides 16:253–263
13. Rundeddu E, Lizarraga JJ, Buscarnera G (2021) Hybrid stochastic-mechanical modeling of precipitation thresholds of shallow landslide initiation. arXiv preprint arXiv:2106.15119
14. Lizarraga JJ, Frattini P, Crosta GB, Buscarnera G (2017) Regional-scale modelling of shallow landslides with different initiation mechanisms: sliding versus liquefaction. Eng Geol 228:346–356
15. Grass G, Rensing C, Solioz M (2011) Metallic copper as an antimicrobial surface. Appl Environ Microbiol 77(5):1541–1547
16. Miao R et al (2019) Online defect recognition of narrow overlap weld based on two-stage recognition model combining continuous wavelet transform and convolutional neural network. Comput Ind 112:103115