

# Image Processing and Machine Learning Methods Applied to Additive Manufactured Composites for Defect Detection and Toolpath Reconstruction



Guan Lin Chen and Nikhil Gupta

## 1 Introduction

Additive manufacturing (AM) plays a crucial role in the fields such as aeronautical, automotive, and medical [1, 2], providing possibilities for low cost parts, highly customizable designs and small production runs [3, 4]. AM relies on a variety of software tools and cloud resources to make it a cyber-physical system [5, 6]. For AM, a wide spectrum of feed materials is available across the entire material usage of polymers, metals, ceramics and composites [7–9]. Parts made of glass and carbon fiber filled polymer matrix composites (PMCs) are now being 3D printed as [10, 11]. It is reported that the parts printed by different tool paths can have different properties because of directionality in material orientation and defects [12]. In case of composite materials, the 3D printing toolpath can be used as a method to orient the fibers in a certain direction and obtain a customized part [13]. AM technologies have improved significantly in recent years; however, there still exist numerous challenges in obtaining high quality, resolution and surface finish required for many applications [14–16]. For example, in powder bed fusion, uniformity in the packing of bed from one layer to the other is important for optimizing the processing parameters, which controls the porosity of the powder bed so that the final part has uniform and maximum density [9]. During fabrication of a complex shaped object using material extrusion methods, which is one of the most commonly used by AM methods [17], an outline is printed first to more precisely define the shapes and then an infill pattern is used to deposit material within the contour's shape [18]. This kind of space filling mechanism leads to gaps at the end of the deposited lines and causes porosity if the process is not well optimized, which affects the mechanical properties of the printed part [19]. Additionally, CAD models are widely used in the AM to demonstrate a

---

G. L. Chen · N. Gupta (✉)

Composite Materials and Mechanics Laboratory, Mechanical and Aerospace Engineering  
Department, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA  
e-mail: [ngupta@nyu.edu](mailto:ngupta@nyu.edu)

digital model and also used for G-code transformation for 3D printers to manufacture the part [20]. However, the curvatures present in the CAD models are often reformed with linear segments in file formats such as STL and can result in the loss of dimensional accuracy in the printed part [5, 21]. These effects cannot be avoided when using material extrusion methods, leading to porosity and micron-sized voids near curvatures, especially in parts of complex geometry and curvatures. The imperfections can be visualized using both destructive and non-destructive imaging methods.

A variety of methods, such as manual or computer assisted non-destructive inspection methods are used to perform quality assessment, which is quite challenging in the AM domain [22], whether during the process or post-processing to maintain high quality as well as the high manufacturing throughput [23, 24]. In AM, parts are printed in several hundred layers, which can not only be used for locating defects but also can be used to hide specific information within layers [25–27]. The layup process of composite manufacturing is somewhat similar to the layer by layer manufacturing of AM and the machine learning (ML) trained model can be applied to composite quality assessment [13, 28]. Artificial Neural Network, which is inspired by biological neural networks, showing efficiency in dealing with intricate nonlinear behavior and has a strong physical foundation for use in the materials science field [29]. ANN has been used in prediction and optimization of material properties [30], especially in the AM domain, which has a strong process–structure–property relationship [31]. ML methods are now being applied to a variety of problems in materials science, including fields such as development of battery materials and testing of composite materials [32, 33] where enormous amounts of raw data are generated on a daily basis that can be used to train the ML models for effective decision-making [34]. ANN is a computational ML method inspired by the architecture of biological neural networks [35] that has been applied in many problems related to composite materials [36–39] and other materials for defect detection [40, 41].

Micro-CT ( $\mu$ CT) scan, which has been used in the medical field for decades, provides a large image database and is finding increasing applications in the characterization of AM parts. Such large image sets are a limitation for manual inspection methods, but an asset for training of ML methods [42–45]. It is beneficial to preprocess the meaningful features that are used for prediction to reduce the training effort and increase the accuracy of the designed ML algorithm. The location and orientation of the reinforcing fibers govern the mechanical response of composite materials, which can be detected by ML methods [46]. ML methods have also been used on optical images to identify and classify two-dimensional materials [47]. The use of ML methods on large image databases sometimes requires significant signal processing effort for making the search faster for implementation in the real time defect detection systems.

In this work, describes in detail the steps required for image processing, ANN model training and validation, and defect detection using a database of images obtained on a 3D printed composite material specimen. A glass fiber reinforced PMC filament is used to fabricate specimens by a commercial material extrusion 3D printer. The specimens are imaged using a  $\mu$ CT scanner. The image dataset is used to train a ML algorithm. Previous studies revealed that 2D images with

irregular outlines increase the difficulty in analyzing the microstructure features of composite materials such as the fiber orientation identification [13, 47], such limitations are overcome by cropping the training images in circular shape during training. Application of image processing methods to reduce the size of the dataset helps in making the training significantly faster. In addition, two kinds of ANNs, namely Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), with three different architectures—one-dimensional and two-dimensional algorithms in CNN, and Long-Short-Term-Memory cell in RNN and its related Python codes are presented, and the implementations of the three trained model and the resulting tests are presented in this work.

## 2 Methods

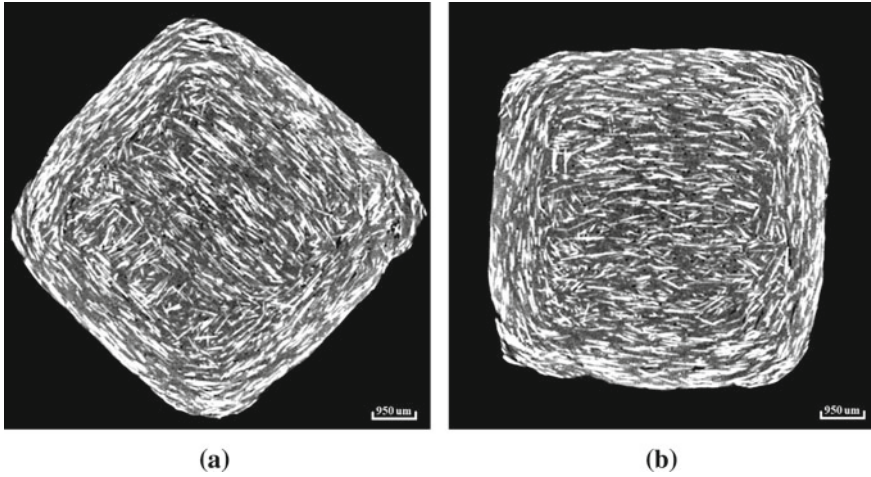
### 2.1 Sample Preparation

The samples are designed using SolidWorks 2017 (Waltham, MA) and saved in STL format. The shape of the sample is a cube with 6 mm side. ABS-GF10 glass fiber reinforced acrylonitrile butadiene styrene (ABS) filament of 1.75 mm diameter, manufactured by 3DXTECH, Grand Rapids Michigan, USA, is used for 3D printing. ReplicatorG software is used for preparing the sliced model and generating the G-code. The printing parameters included 100% solid infill, feed rate of 41 mm/s, extrusion temperature of 220 °C, and build platform temperature of 120 °C. A FlashForge-Creator Pro Dual Extruder printer is used for printing the specimens.

### 2.2 $\mu$ CT Scan Image

A Skyscan 1127 (Bruker, Belgium)  $\mu$ CT scan system is used at source voltage of 44 kV, current of 222  $\mu$ A and the camera pixel size of 9.5  $\mu$ m with a rotation step of 0.6° per scan, for 360° rotation. Normal scanning time is about 40 min with lowest resolution. However, larger specimens can take 6–8 h in one scan based on the image resolution. Each scan generates thousands of images, which are usually saved in tiff format.

The images acquired from  $\mu$ CT scan were reconstructed using NRecon software. In this step, the image can be tuned with function such as “smooth”, which can improve the clarity of the image slice, “contrast”, which can reduce the noise and “rotation”, which is used to rotate at a fixed angle to output the reconstructed images with desired angle as Fig. 1 shown. Since, the sample was randomly mounted on the specimen plate and the scanning process involved rotation, the printing direction of the output images was random with respect to the scanning direction. It is important to designate a reference angle so that all the images can be interpreted with respect



**Fig. 1** An example of a  $\mu$ CT scan output image with **a** random angle and **b** after rotation to  $0^\circ$

to the same reference framework. Since fibers are included in the feed material and the fibers provide high contrast, they are used for identification of the direction. In this work,  $0^\circ$  is defined as horizontal reference. Therefore, the reconstructed images are rotated using the “rotation” function in NRecon software to output the images to be the reference images of  $0^\circ$ . Once the images are rotated, they can be directly compared. The present work is exploratory for developing the image processing and ML model training methods; hence, the geometry of the 3D printed part is kept simple.

The square shape of the specimen having four sharp corners is a consideration in this work. The sharp corners would inevitably affect the accuracy for ML because any sharp angles in the image may cause bias while training the model, resulting in inaccurate predictions. To increase the accuracy of the trained model, the data selection and features extraction are the key steps. Therefore, it is better to sub-segment each  $\mu$ CT scan image slice into smaller images and use circular shape sub-images. Rotation of circular sub-images will preserve the features in the circular area without any interference at the edges. The model trained with a circular image dataset is still capable of predicting a square image. However, if the model is trained on a square image dataset, it will not be able to predict a circle image with accuracy.

### ***2.3 Circular Image Dataset Preparation for ML Model Training***

Two prediction purposes are introduced. One is to build a ML model capable of predicting the printing direction and its angular information in each layer of the

sample, which can then provide the information of the printing angles used in the 3D printer. Another purpose is to predict the movement of the print head in each layer, which can then be used to analyze the printing signature throughout the printed sample.

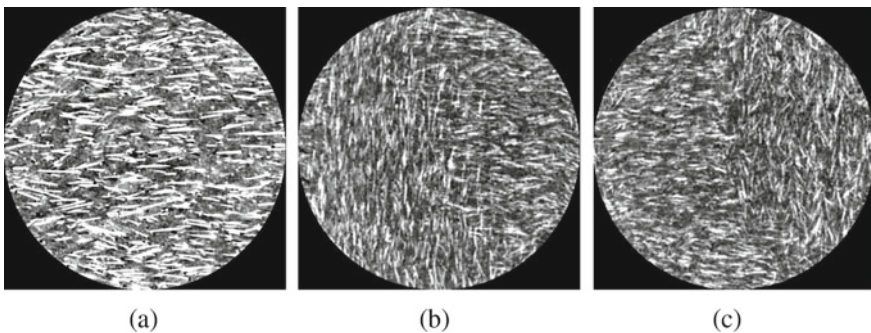
There are two different methods to obtain the required dataset. The first method uses image reconstruction after the  $\mu$ CT Scan and the second method uses a Matlab code. Both of the methods will be discussed thoroughly in the following section.

## 2.4 Dataset for Orientation Prediction in Each Layer

In this section, the prediction of the overall printing orientation in each layer of the sample is developed. Circular sub-images are first extracted from each image slice as shown in Fig. 2. The function of “region of interest” is used during the image reconstruction with NRecon software to capture a certain area of the scanned image. Figure 2a shows an image where glass fibers seem to be oriented in the horizontal direction. Figure 2b, c show two sub-images that contain fibers oriented in multiple directions. These features appear from two different slices because the scan resolution is much finer than the deposited layer thickness and the specimen shows some warpage due to thermal contraction upon cooling. Such intermixing of features may also take place due to mismatch between the printing plane and CT scanning plane.

Depending on the resolution setup for the  $\mu$ CT scan and the specimen size, the total numbers of images will be different in each dataset. For the specific sample used in this work, there are 663 images obtained after the  $\mu$ CT scan.

Next, a reference angle needs to be decided. In this work, images with horizontal glass fiber are designated as  $0^\circ$ . For training the ML algorithm, a sufficiently large database is necessary. Moreover, to distinguish the fiber angle, the model needs to be able to predict fibers along any angle in the entire  $360^\circ$ . However, it actually only needs to predict  $180^\circ$  to indicate the whole  $360^\circ$ , since  $10^\circ$  can represent  $190^\circ$  and



**Fig. 2** Sub-images extracted from a  $\mu$ CT scan image slice

$0^\circ$  is the same as  $180^\circ$ . Therefore, in order to train the model, images with clear visibility of  $0^\circ$  fiber angle are synthetically rotated counterclockwise at intervals of  $1^\circ$  each from  $0^\circ$  to  $180^\circ$  using a Python code shown as Fig. 3 to get the images database to train the model for predicting various rotation angles. The rotation of the images can be conducted at smaller steps but the discontinuous short fibers in the present case are not oriented exactly in the same direction. Some local variations exist within one extruded line as well as some fibers are bent. Such conditions will provide false positives. Hence, the rotation step is kept at a higher angle. The process allows creating a dataset with controlled orientation of the glass fiber to train a model for printing orientation prediction in each layer.

A few  $\mu$ CT images showing overlapping printing directions in Fig. 2b, c are mostly at the boundary of two deposited layers due to the slight slope that can be either in the printing or in the specimen positioning on the  $\mu$ CT stage. These images are removed from the dataset in the present case because there is enough information available from other slices that are cleaner. In order to have a defined/labeled angle, a set of 207 images that show clear orientation of fibers is identified and rotated so that the fiber orientation is horizontal. These images are labeled as  $0^\circ$  fiber orientations. All the images are cropped with a region of interest to decrease the number of pixels and remove bias caused by the sharp angles in four corners for machine learning training. The cropped images, shown in Fig. 2, have a pixel size of  $536 \times 536$ . Individual fibers in each image show some variation in their direction. However, the algorithm takes the global signature as the features for the  $0^\circ$  and disregards the individual fiber orientation. Each of these images are then rotated counterclockwise from  $0^\circ$  to  $180^\circ$  at  $1^\circ$  interval using a Python code. This procedure allows creating a large training database with controlled fiber orientation and trains the model to identify any angle. This procedure resulted in 37,467 images. The process of synthetically rotating the images and cropping them with built-in function during image reconstruction with

```

1. # Rotate the Images and save to different dir
2. from PIL import Image
3. import matplotlib.pyplot as plt
4. import os
5. import shutil
6. path = 'C:/Users/User/Desktop/Guanlin_CNN1D/CNN1D/2D book 207 testing' # The path of 0
   deg reference images
7. for deg in range(181): # Save rotated image into folder named 0, 1, 2, 3,...,180
8.     out_put = 'C:/Users/User/Desktop/Guanlin_CNN1D/CNN1D/0_180 book testing/'+str(deg)
9.     print(out_put)
10.    for image in os.listdir(path):
11.        img = Image.open(path+'/'+image)
12.        img.rotate(deg).save(image)

```

**Fig. 3** Python code for rotating image form  $0^\circ$  to  $180^\circ$  at  $1^\circ$  interval and save to folders named as 1, 2, 3...180

NRecon software used to generate training data is known as image augmentation. This process is helpful in training the ANN model to be robust.

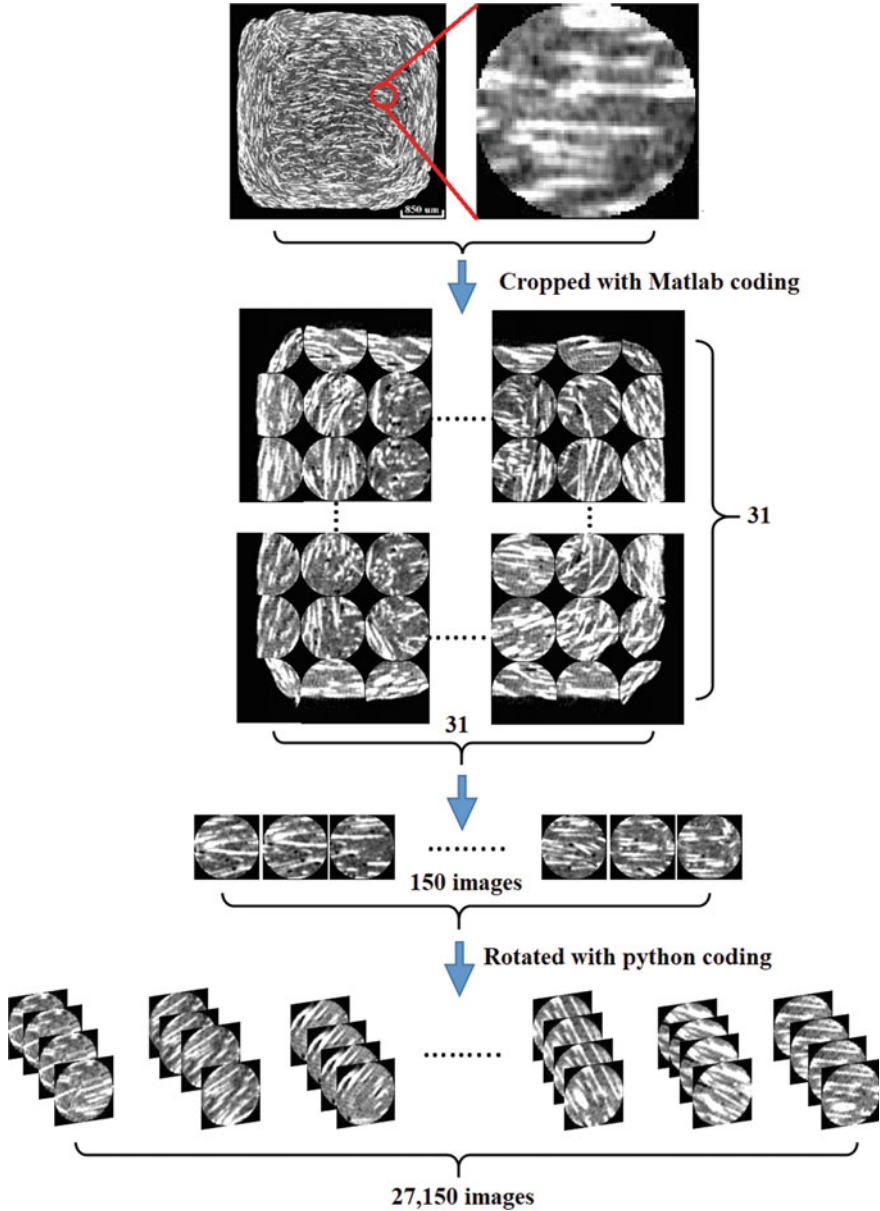
### 2.4.1 Dataset for Toolpath Prediction in Each Layer

In this section, training dataset preparation is slightly different compared to the previous section. Here, the purpose is to determine the printing path, or toolpath, of each layer of the sample with specific image features for the model training. Moreover, the algorithm used for this dataset is 1D and RNN, which needs to transfer the features contained in a 2D image into a meaningful dataset with 1 dimensional format. Therefore, the training dataset should contain meaningful features, which represent the movements of the print head. Thus, one layer of image with clear fiber orientation is used to conduct the feature extraction for model training dataset preparation. Here, instead of using region of interest to extract the features from a certain area, for toolpath prediction, the whole image slice is preferable since the toolpath of the whole layer and the whole sample need to be predicted to completely recreate the toolpath of the sample.

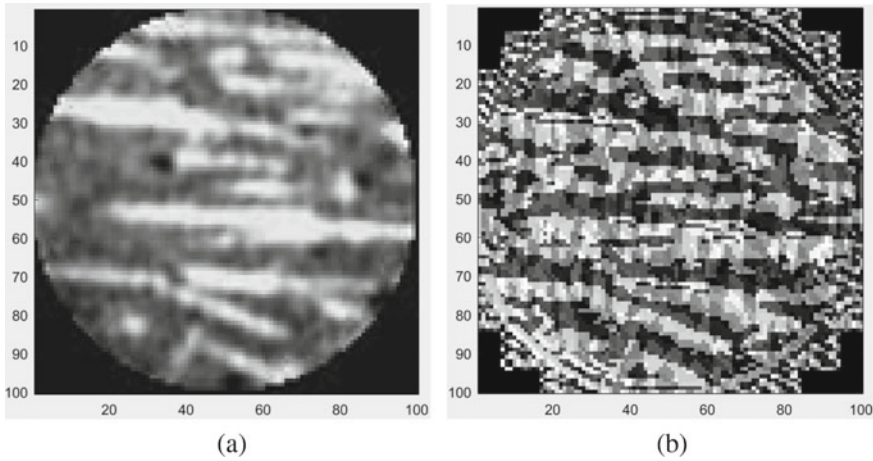
Each  $\mu$ CT image slice represents a variety of fiber directions based on the space filling algorithm used for determining the print head movement. The ML model tuning dataset needs to identify these features, Hence, each image is sliced in circular sub-images of  $100 \times 100$  pixels. The sub-sampling process and the resulting images are shown as Fig. 4. This process results in 961 cropped circular images for each  $\mu$ CT image slice. The circular shape helps in reducing the error caused by irregular outline of the specimen. In this work, 150 images with clear visibility of  $0^\circ$  fiber angle are selected and then the images are synthetically rotated counterclockwise at intervals of  $1^\circ$  each from  $0$  to  $180^\circ$  using a Python code shown as Fig. 3. Similar to the previous section, this cropped circular images for each  $\mu$ CT image slice help in reducing the error caused by irregular outline of the specimen. Then, 5 images are randomly selected to be the test image set, 115 images are randomly chosen to be the training data and the rest 30 images are used to be the validation images. Three of them are then rotated with Python code to create the training, validation, and test dataset with total images of 27,150 images for model training and testing. A special image lossless-processing algorithm called BSIF then used to convert the 2D images into a 1D numerical dataset for reducing the computing power and increasing the processing speed. The detail of the BSIF process are discussed in the next section.

### 2.4.2 Binarized Statistical Image Features (BSIF) Algorithm

Binarized Statistical Image Features (BSIF) algorithm is used to convert an image into a binary image format without losing valuable features [48]. Figure 5 shows an example of the image produced by the BSIF algorithm. The images processed through BSIF algorithm are used for training, validation and testing the ML algorithm. The output image is a binary code for each pixel in the image and is stored in a format



**Fig. 4** The image dataset (27,150 segmented images) obtained from  $\mu$ CT scan after removing overlapping images and being labeled as  $0^\circ$  to  $180^\circ$  for ML



**Fig. 5** BSIF representation of the glass fiber orientation image with  $100 \times 100$  pixels. **a** The original circular cropped 2D image and **b** the 2D image converted by BSIF

of 1D numerical array, which makes it convenient to handle large amounts of data. Although the visual representation of features in Fig. 5 is not well resolved, such images are proven to perform well in ML. Use of BSIF code reduced each image to  $1 \times 256$  numerical data as shown in Table 1, which exponentially reduces the computational expense involved in running the ML algorithm. Since 27,150 images are used, the resulting data was saved as a csv file with 27,150 rows of features, which is extremely helpful for ML. This procedure uses a large dataset but the training time for the ANN is significantly reduced without losing the accuracy for interpreting the features present in the image [48]. The 2D images are converted into 1D numerical data with one row and 256 columns and since the more meaningful features represent an output the higher accuracy of the model can be obtained.

In this work, only the dataset for toolpath prediction in each layer (introduced in Sect. 2.4.1) needs to be converted with BSIF since the 2D CNN needs original 2D images as its input data.

## 2.5 Machine Learning

Two kinds of ANN architectures are used to build models using Python with Tensorflow: RNN and CNN. Both neural networks can deal with sequences with variable lengths. The RNN uses memory-state to process the input data on each neuron. In this work, an RNN architecture with 5 layers and 64 Long Short-Term Memory (LSTM) cells is used to train the model. A CNN model is trained with total 5 layers and kernel sizes 5 and filter size 80 in the first layer and the second layer to iterate through the data to train the model.

**Table 1** Image data converted into 1 row of numerical data after BSIF process

|        |          |          |         |          |     |         |          |          |          |         |
|--------|----------|----------|---------|----------|-----|---------|----------|----------|----------|---------|
| 0      | 1        | 2        | 3       | 4        | ... | 251     | 252      | 253      | 254      | 255     |
| 0.1832 | 0.003904 | 0.001528 | 0.00112 | 0.004256 | ... | 0.00428 | 0.001108 | 0.001192 | 0.001904 | 0.00552 |

1 Row  $\times$  256 Columns

The original images are used for 2D CNN algorithm because it has its own image conversion process built in the algorithm. On the other hand, for 1D CNN and RNN, the images converted with BSIF are loaded and rearranged in the form of an array as (20,815, 1, 256) for training feature. This implies that this training set has 20,815 data representing the true angle label, with time-step as 1, and 256 features points. The same process applies to validation dataset, which results in an array of (5430, 1, 256) and the test dataset has the numpy shape of (905, 1, 256) for RNN algorithm and (20,815, 256, 1), (5430, 256, 1), (905, 256, 1) for training, validation and testing, respectively, in 1D CNN algorithm. The loss function depends on the desired result parameter such as the fiber angle. Hence, the mean square error (MSE) was used in both RNN and 1D CNN as the loss function in the model. The predicted values and the actual value of the angle are used to calculate MSE.

In summary, this work uses three different ML algorithms. 1D and 2D CNN and RNN are compared for their accuracy. Each of them target different dataset and loss function, the computer language used is python platform with Tensorflow in Windows 10 environment.

## ***2.6 The Architectures of the Machine Learning Algorithms***

### **2.6.1 2-Dimensional Convolutional Neural Network (2D CNN)**

The 2D CNN ML algorithm uses Python platform, with Tensorflow, CNN (Convolutional Neural Network). CNN works well for identifying simple patterns within the data, which are then used to form more complex patterns within higher layers. CNN is very effective in deriving interesting features from shorter (fixed-length) segments of the overall data set and where the location of the feature within the segment is not of high relevance.

In 2D CNN algorithm, the input is the original image data and the ML algorithm convert the image features in its own system. The loss function in 2D CNN is “sparse categorical cross entropy”, which is used to distinguish different categories. In 2D CNN, the model can be used to predict the printing orientation in each layer as a classifier. Hence the predicted values represent the orientation of each layer, and can iterated through whole sample to provide the information of the printed direction. However, it is not capable of identifying the toolpath in each layer. Also, since it uses the original images as the input, the processing needs a huge computing power. Additionally, when increasing the number of categories, the training time increases substantially. Since the process uses original images and the in-built image processor provides a lossless conversion procedure, the accuracy can reach almost 99% with proper setup parameters. Thus, the 2D CNN is good at image analysis, especially if high performance computing facilities are required. In this section, a 2D CNN model and the process steps will be presented in detail.

The dataset for orientation prediction in each layer (Sect. 2.4), which contains 207 clear views of 0° orientation images is used to create the input data. The images

need to be saved in folders according to the angle they represent, which means each classification category is saved in corresponding folder. The  $0^\circ$  reference images are split into 3 subsets of images, one for training, one for validation and one for testing. Here, 7 images are chosen to be the test data, and the ratio of training and validation data is set as 8:2. Thus, the rest of the 200 images are split into 166 images for training and 34 images for validation. With the help of python code, as Fig. 3 shows, the split images are rotated from  $0^\circ$  to  $180^\circ$  with  $0^\circ$  interval and saved to folders named as 1, 2, 3...180 for training, and the same process is conducted to create validation dataset and test dataset. By doing so, the training dataset with 30,046 images and the validation dataset with 6154 images and the test dataset with 1267 images are obtained and each angle of images is saved to the corresponding folder.

Once the training, validation and test dataset are prepared, they are fed into the 2D CNN algorithm for model building. The image dataset needs to be re-sized from  $536 \times 536$  to  $100 \times 100$  pixel to reduce the needed computing power and then create the training and validation datasets. Last, the training and validation datasets need to be appended to corresponding features and labels, which are then converted into numpy arrays with the shape of (30,046, 100, 100, 1) for training dataset, (6154, 100, 100, 1) for validation dataset in order to fit the 2D CNN model training algorithm. The first number represents the number of images, the second and the third numbers represent the pixel size in  $x$  and  $y$  directions, and the fourth number represents the number of images in each sequence. The Python code used for the 2D CNN image processing is shown as Fig. 6.

The architecture of the 2D CNN algorithm is shown as Fig. 7, which has 5 layers. The filter size used is 80, and the kernel size in layer 1 is (5, 5) and (3, 3) in layer 2. The Maxpooling size is (2, 2) in layer 1 and 2. The hidden layer Dense is 200. The output layer has 181 categories, so output Dense is set as 181. The activation is selected as "softmax". Batch size iterated is 100, and the epochs is set as 50. Also, a Dropout function of 0.1 is used to intentionally drop 10% of the training data in each training epoch to prevent the overfitting.

The setup parameters used in this 2D CNN algorithm are obtained by trial-and-error and the parameters are extremely data-dependent. Therefore, different numbers of image, different pixel size of image, and even different shapes of the image used for training will need to find the suitable parameters accordingly. A checkpoint function is used to monitor the accuracy during each epoch, and save the best model throughout the whole training process. The total time for the training is 24 h.

### 3 2D CNN Result

When the training process is completed, first thing is to check the fitting status during the training process, whether it is overfitting or underfitting. To do that, a recall function in Python is used and the training and validation histories are plotted as Fig. 8. The result in this training process shows no overfitting or underfitting, since the validation result and the training result have the same trend and do not show large

```

1. # Resize the image to speed up the process
2. Img_Size = 100
3. # Training_dataset preparation
4. training_data = []
5. def create_training_data():
6.     for category in CATEGORIES:
7.         path = os.path.join(DATADIR, category)
8.         class_num = CATEGORIES.index(category)
9.         for img in os.listdir(path):
10.            img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
11.            new_array = cv2.resize(img_array, (Img_Size, Img_Size))
12.            training_data.append([new_array, class_num])
13. create_training_data()
14. # Validating_dataset preparation
15. validating_data = []
16. def create_validating_data():
17.     for category in CATEGORIES:
18.         path = os.path.join(DATA_valid, category)
19.         class_num = CATEGORIES.index(category)
20.         for img in os.listdir(path):
21.            img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
22.            new_array = cv2.resize(img_array, (Img_Size, Img_Size))
23.            validating_data.append([new_array, class_num])
24. create_validating_data()
25. # Re-shape the Training dataset
26. X=[]
27. y = []
28. for features, label in training_data:
29.     X.append(features)
30.     y.append(label)
31. X = np.array(X).reshape(-1, Img_Size, Img_Size, 1)
32. y = np.array(y).reshape(-1, Img_Size, Img_Size, 1)
33. print('X shape :'+str(X.shape))
34. # Re-shape the Validating dataset
35. X_valid = []
36. y_valid = []
37. for features, label in validating_data:
38.     X_valid.append(features)
39.     y_valid.append(label)
40. X_valid = np.array(X_valid).reshape(-1, Img_Size, Img_Size, 1)
41. y_valid = np.array(y_valid).reshape(-1, Img_Size, Img_Size, 1)

```

**Fig. 6** The python code used for resizing images and dataset preparation in 2D CNN algorithm

```

1. # 2D Convolutional Neural Network architecture
2. X = X/255
3. X_valid = X_valid/255
4. model = Sequential()
5. # Layer 1
6. model.add(Conv2D(80,(5,5), input_shape = X.shape[1:], padding = 'SAME'))
7. model.add(Activation("relu"))
8. model.add(MaxPooling2D(pool_size=(2,2)))
9. # Layer 2
10. model.add(Conv2D(80,(3,3), padding = 'SAME'))
11. model.add(Activation("relu"))
12. model.add(MaxPooling2D(pool_size=(2,2)))
13. # Layer 3
14. model.add(Dropout(0.1))
15. model.add(Flatten())
16. # Hidden Layer
17. model.add(Dense(200))
18. model.add(Activation('relu'))
19. # Output Layer
20. model.add(Dense(181, activation='softmax'))
21. model.compile(loss='sparse_categorical_crossentropy',optimizer="adam",metrics=['accuracy'])
22. filepath = 'C:/Users/User/Desktop/2D CNN model-{epoch:02d}-{val_acc:.3f}.model'
23. checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=2, save_best_only=True,
    mode='auto')
24. model.summary()
25. history = model.fit(X, y, batch_size = 100, epochs=50, verbose=1,
26.                    validation_data = (X_valid, y_valid), callbacks=[checkpoint])

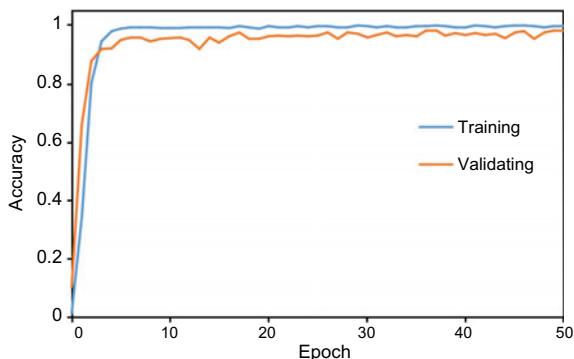
```

**Fig. 7** 2D CNN architecture for the classification model

variation. If the graph shows a large gap between the two lines, it indicates that the model is not trained properly and the setup parameters need to be further tuned. The acceptability of the result also depends on the desired accuracy. Typically, in 2D CNN machine learning, the accuracy of training is often higher than the accuracy of validation, which is also observed from the resulting plot shown in Fig. 8.

To test the accuracy of the trained model, the test dataset is needed, which has been separately prepared with 1267 data using the same preparation process as the training dataset. An array shape of (1267, 100, 100, 1) is required to be compatible with the training dataset. In this 2D CNN training, the shape used for training is (30,046, 100, 100, 1). Therefore, the test dataset must have the same pixel size of  $100 \times 100$ . Then the folders with images are loaded in the Python code, and the output categories, which contains  $0^\circ$  to  $180^\circ$ , are created. A function “model.evaluate” is used to check

**Fig. 8** Training and validation history in each epoch for 2D CNN model



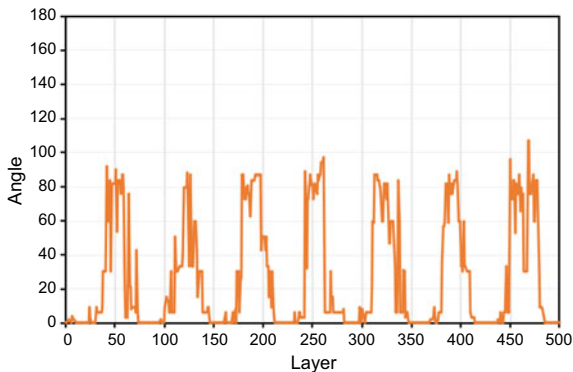
the model accuracy. By doing so, the trained model with the accuracy of 0.9684 is achieved, which can be used in a practical situation.

Next, the prediction accuracy of the trained model is determined by using a set of images of unknown angular information. Here, another 3D printed specimen of a similar type is imaged by  $\mu$ CT Scan and 500 images with unknown angular information are obtained. The images are then renamed according to its layer height from the bottom of the sample to its top with the name of “Layer 1”, “Layer 2”, etc. These 500 images are then saved in a folder and then loaded into Python code. Next, the images are used to create a test dataset and converted into a numpy array with the shape of (500, 100, 100, 1). The names of the images are appended to the “Layer” column, and the converted images’ features are used for prediction. The results are appended to the “Predicted Angle” column. Last, the outcome is saved in the tabular form and the predicted angle distribution for each layer is plotted, as shown as Fig. 9, which can reveal the printing orientation of the tested sample for each layer. Here, the predicted angles are located around  $90^\circ$  and  $0^\circ$  showing that the sample is printed with these two angles repetitively. One major benefit is that the prediction process is very fast. The dataset has 500 images and the time taken for prediction is less than a second. Therefore, even though it takes 24 h to train the model, the time required for prediction is extremely short.

### 3.1 1-Dimensional Convolutional Neural Network (1D CNN)

In this section, a 1-D ML algorithm is introduced. The 1D CNN, which means the input data is 1-Dimensional. Therefore, 2D images must go through the BSIF algorithm to be converted into 1D data. In the 1D CNN algorithm, a loss function, Mean Square Error (MSE) is used. Each image features are converted with the BSIF lossless-process into a numerical data with 1 row and 256 columns, then in order to easily extract the needed features and to designate the angle/label, a column, named “Image Angle” is added as the first column to designate the angle/label and

**Fig. 9** Angular distribution of the tested sample in 500 layers



“Feature\_1”, “Feature\_2”, all the way to “Feature\_256” are add at the top of each feature. Thus, all the information representing each image in a fixed-length and its angle/label and features are saved as a CSV file.

The 1D CNN algorithm is capable of predicting the printing orientation in each layer. It is also capable of predicting the toolpath of the sample in each layer. The database containing 27,150 images needs to be divided into training, validation, and test datasets. Since 150 images with clear view of  $0^\circ$  fiber orientation are selected as described previously, the images are used to rotate  $1^\circ$  interval counterclockwise from  $0^\circ$  to  $180^\circ$  to represent 180 angles. Moreover, in order to avoid bias during the model training, 115 images are picked to use as the training dataset for each angle, which total to be 20,815 sub-images. Then 20 images from each angle are added to the validation dataset, which sums up to 5430 sub-images. Finally, 905 sub-images are used in the test dataset. By feeding the dataset with evenly distributed weights, the ANN will have the least bias when the training is completed. In 1D CNN, the loss function is “Mean Square Error”, and the output is a value representing the predicted angle. The CSV files for training data, validation data and test data are loaded using the “panda” function in Python to train the model.

The dataset needs to be prepared to form a correct format after being loaded in order to feed into CNN algorithm. The Python code used is shown as Fig. 10, where “data.values” function is used to capture the features within the data. The first colon in the bracket means the function iterates through every row’s data, and the number after the comma represents the data in each number of the column. Here, “2:” means obtaining the data from the second column all the way to the last column and then the value is used as training features. Then, the output labels need to be defined. “[:, 1]” is used to obtain the data in the first column for every row and the value is defined as training labels. The same process is used for the validation dataset to obtain the validation features and labels.

After the process of iterating through every row and column of the CSV file, a re-shape function is used, which creates a dataset with a shape of (20,815, 256, 1) for training feature dataset, (5430, 256, 1) for validation feature dataset and (905, 256, 1) for testing feature dataset. Here, the first number represents the number of the

```

1. # Training and Validating Features and Labels Designation
2. X_train = data.values[:, 2:]
3. Y_train = data.values[:, 1]
4. X_valid = val_data.values[:, 2:]
5. Y_valid = val_data.values[:, 1]
6. # Reshape the nparray to fit the training and testing model
7. X_train = np.array(X_train, dtype=np.float).reshape(X_train.shape[0],X_train.shape[1],1)
8. Y_train= np.array(Y_train, dtype=np.float).reshape(Y_train.shape[0],1)
9. X_valid = np.array(X_valid, dtype=np.float).reshape(X_valid.shape[0],X_valid.shape[1],1)
10. Y_valid= np.array(Y_valid, dtype=np.float).reshape(Y_valid.shape[0],1)

```

**Fig. 10** Python code used to prepare the training and validation dataset in 1D CNN

data/images, the second number represents the features each data has, and the third number represents the number of images for each process. As for training, validation and test label datasets, the shape of the array is (20,815, 1), (5430, 1) and (905, 1) respectively. The first number represents the number of label data, and the second number represents the number of images for each process.

For the output of an angle prediction, it would be easier to understand when the output is a value that represents the predicted angle. Hence, the loss function “Mean Square Error” (MSE) is used. The model can output a number value to be the representation of an angle for the input feature.

The architecture of the 1D CNN is shown as Fig. 11. Similar to 2D CNN architecture, the checkpoint function is used to capture the model with the lowest MSE value throughout the training process. There are 5 layers used in the 1D CNN architecture. The filter size for layer 1 and 2 is 256, which means the whole 256 features are considered, the kernel size is 1, which means 1 full image features are processed at each time. Padding used is “same”, the activation used in each layer is “relu”, the hidden layer has Dense of 200 and the loss function is “Mean Square Error”, the optimizer is Adam, which provides a learning rate of 0.001, the batch size is set as 128, and total 10,000 epochs is used. The verbose in checkpoint is set as 2, which means the process will just mention the number of epochs, and in “model\_m.fit” the verbose is set as 1, which will show an animated progress bar for the user to observe if the model is properly trained. The parameter setup in this training is also extremely data-dependent, therefore, with different datasets, the parameters need to be tuned again.

### 3.2 1D CNN Result

After completion of the model training, the first thing is to check the training history to make sure it has no overfitting, which shows the validation curve to be much lower than the training curve or underfitting, which shows the training curve to be much

```

1. # 1D CNN Convolutional neural network architecture
2. model_m = Sequential()
3. # Layer 1
4. model_m.add(Conv1D(256, (1), activation='relu',
5.                     input_shape=(X_valid.shape[1],X_valid.shape[2]), padding = 'same'))
6. # Layer 2
7. model_m.add(Conv1D(256, (1), activation='relu', padding = 'same'))
8. # Layer 3
9. model_m.add(Flatten())
10. # Layer 4
11. model_m.add(Dense(200, activation='relu'))
12. # Layer 5
13. model_m.add(Dense(1, activation='relu'))
14. model_m.summary()
15. filepath = 'C:/Users/User/Desktop/models/1D CNN book-{epoch:02d}-{val_loss:.3f}.model'
16. checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=2,
17.                               save_best_only=True, mode='auto')
18. model_m.compile(loss= 'mean_squared_error' , optimizer= 'adam' , metrics=[ 'mse' ])
19. history = model_m.fit(X_train, Y_train, batch_size=128, epochs = 1000, verbose=1,
20.                       validation_data = (X_valid, Y_valid), callbacks=[checkpoint])

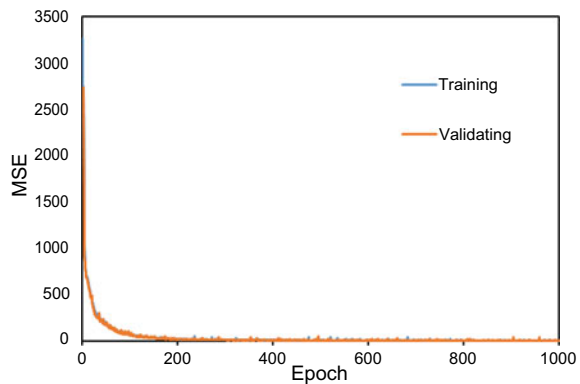
```

**Fig. 11** The ML architecture with 5 layer in the 1D CNN algorithm

lower than the validation curve. The training and validation histories are plotted as Fig. 12. The graph shows no sign of overfitting or underfitting. Then, the function “model.evaluate” is used to test the accuracy of the model. Since the MSE loss function is used, the output of the evaluation is the MSE of the prediction. A MSE of 0.9651 is obtained showing a promising result for the CNN implementation.

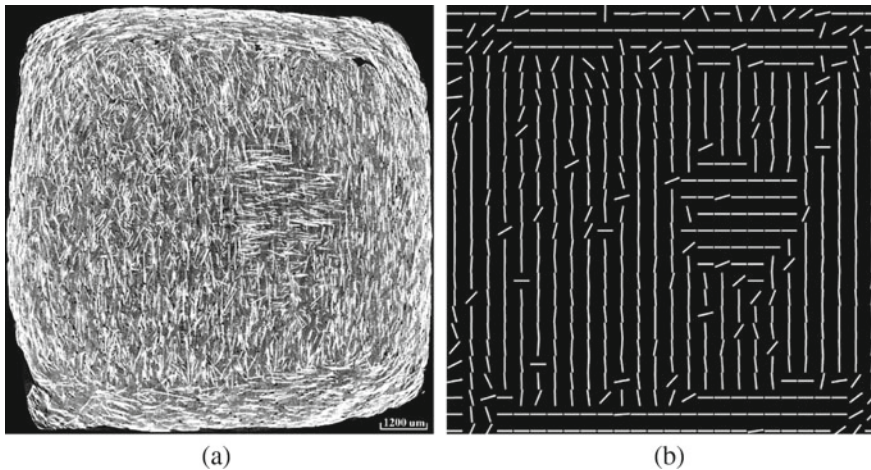
The model trained in this section is used to predict the toolpath of the composite material specimen image set obtained from  $\mu$ CT scan. In this work, the sample used

**Fig. 12** The training and validation MSE of each epoch in 1D CNN model



for toolpath prediction is a square cube with 1 cm length on each side. To predict the toolpath of the whole sample, a  $\mu$ CT scan needs to be conducted to obtain the sliced images. The  $\mu$ CT scan image used to perform the toolpath prediction has pixel size of  $2657 \times 2689$ . Next, similar to all training dataset preparation, the images are cropped as circles using a Matlab code and after the cropping process, 676 images with pixel size of  $100 \times 100$  are obtained. These 676 images are then converted with BSIF and become 676 data. Each data has 256 numerical features and are saved as a CSV file. Thus, a shape of (676, 256, 1) numpy data array is ready for toolpath prediction.

In order to clearly view the toolpath in each layer, a direction indicator is used. The idea is to impose each small cropped circular image with a direction indicator corresponding to its prediction and then combine all the direction indicators to reconstruct the whole image to represent the layer showing the toolpath. Hence, the predicted result needs to be recorded and saved as a CSV file, which has a column showing the region the cropped image belongs to and a column for prediction result of that region. The direction indicator images showing the angle from  $0$  to  $180^\circ$  are saved with a file name with respect to its angle. Then, according to the prediction result, a Python code “if and else” is used to match the prediction result and the corresponding image name of the direction indicator. For example, an image representing region 10 is predicted as  $44^\circ$ , then the Python code will match the prediction result  $44^\circ$  to the direction indicator, which is named as 44 and the direction indicator is saved to represent the region 10. After the process iterates through all predictions, a collection of all 676 direction indicator images is saved. Finally, a  $26 \times 26$  grid is created to display all 676 predicted direction indicators are shown as Fig. 13.



**Fig. 13** Imposed toolpath reconstruction with 1D CNN model

### 3.2.1 Recurrent Neural Network (RNN)

Recurrent neural network (RNN) is a supervised ML algorithm, which is designed to model sequential data. The order is very important in sequential data. There are different forms of sequence modeling algorithms but the one used here is the many-to-one sequence model, which implies that the input data is a sequence but the output is not a sequence, rather a fixed-size vector. In RNN, the hidden layer has inputs from both the input layer and the hidden layer from the previous step. The flow of information in an RNN from one time-step to another introduces memory of past inputs into the network [49].

The algorithm used here is a multilayer RNN, which is used to predict the direction of fibers in a  $\mu$ CT scan image. The input is the  $\mu$ CT-scan image features obtained after the BSIF process and the output is the fiber orientation angle. At any time instance the model uses the information from the past and input to predict the output. Since AM follows a sequential process of printing, the fiber orientation at each layer can be helpful to predict the orientation of fiber of the next layer. Typically, a backpropagation through time (BPTT) algorithm is used to train an RNN, which sometimes has a problem of vanishing gradient. RNN model faces difficulty in learning the long-term dependencies because it is trained with sequential data, which implies that the model will not be able to relate the images which are captured several time steps apart. To address these issues, the RNN architecture with LSTM network is used [50]. RNN with hidden layers containing LSTM cells takes information from the input and from the previous hidden layers and calculates the output through a set of equations and then sends the information to the next layer in the model and to the hidden layer, namely, another LSTM cell in the next time step. LSTM cells are designed to handle the problem of vanishing gradients. These cells have inbuilt default units programmed to remember the updates from the previous time steps without loss of information over long time steps, making them suitable for large image datasets.

As mentioned previously, the BSIF process has converted features of each image into 1 row and 256 columns. A column named “Image Angle” is added to the file as the first column to designate the angle/label. Thus, all the information to represent each image is available in a single row. Any change to the input data affects the algorithm and the setup parameters need to be tuned. In RNN, the dataset for toolpath prediction in each layer is used, which means a model capable of predicting the toolpath can be acquired. Here, the same training, validation and test data of CSV files as 1D convolutional neural network is used. The Python code used for reshaping is similar as the one used in 1D CNN algorithm, where the only difference is in array arrangement. In RNN architecture, the shape of the training data is (20,815, 1, 256), the validation data is (5430, 1, 256), and the test data is (905, 1, 256). The first number represents the number of dataset the CSV file has, the second number represents the amount of data being iterated in each time step, in this case, 1 image is used in a time step. The architecture of the RNN Python code used is shown in Fig. 14. There are 5 layers in the algorithm. The number of LSTM cells used in first 2 layers is 64, the hidden layer has “Dense” of 128 with only 1 output value, the loss function is “MSE”, the optimizer is Adam, which has a learning rate of 0.001 and the batch size

```

1. # RNN Recurrent Neural Network architecture
2. model = Sequential()
3. # Layer 1
4. model.add(LSTM(64, input_shape=(X_train.shape[1:]), return_sequences=True))
5. model.add(BatchNormalization())
6. # Layer 2
7. model.add(LSTM(64, return_sequences=True))
8. model.add(BatchNormalization())
9. # Layer 3
10. model.add(LSTM(64,))
11. model.add(BatchNormalization())
12. # Hidden Layer
13. model.add(Dense(128, activation='relu'))
14. # Output layer
15. model.add(Dense(1))
16. model.summary()
17. # learning rate 0.001
18. model.compile(loss='mse', optimizer='Adam' , metrics=['mse'])
19. filepath = 'C:/Users/User/Desktop/models/RNN book-{epoch:02d}-{val_loss:.3f}.model'
20. checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=2,
                               save_best_only=True, mode='auto')
21. history = model.fit(X_train, Y_train, batch_size = 128, epochs=10000, verbose=1,
22.                    validation_data = (X_valid, Y_valid), callbacks=[checkpoint])

```

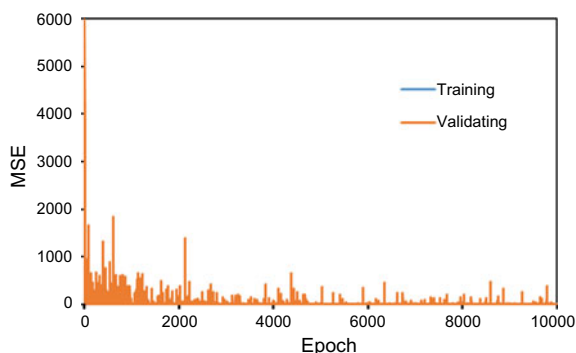
**Fig. 14** The architecture of the RNN machine learning of python coding

is 128, and epochs are 10,000. Similarly, a checkpoint function is used to capture the best model. Since the model is trained repeatedly, and the accuracy of the last trained model does not guarantee to be the best one. Most of the times, the model reaches its peak performance, but would be overwritten by the next trained model. The callbacks function can monitor the validation loss of each epoch and then save the model that is more accurate than the previous trained model.

### 3.2.2 RNN Result

A similar testing procedure as the 1D CNN are applied here for RNN model accuracy checking. The recorded training and validation histories are plotted and shown as Fig. 15, which shows no sign of overfitting or underfitting. The deviation of RNN is relatively greater than the CNN method, and this is why a checkpoint function is necessary in RNN method for capturing the best trained model. The test MSE in this training is 0.059, which indicates the performance of the model is good to predict the toolpath of the sample's layer.

**Fig. 15** MSE history of training and testing for each epoch in RNN model

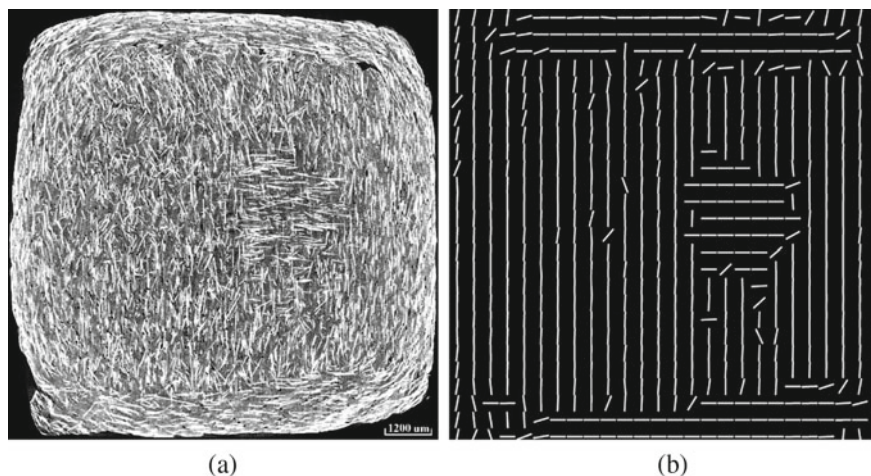


To implement the model, the data prepared for 1D CNN are used. 676 cropped circular images converted with BSIF process, which become 676 rows with 256 features for each image, are saved as CSV files. Then, the CSV file is loaded into Python with the “panda” function. Since the purpose is to predict the angular information for each cropped circular image, 256 features are used as input data and undergo a similar data preparation as the one used for the training dataset, except, there is no output label in these 676 images. After the prediction, a set of values representing the angular information is extracted and saved as a CSV file.

The same method of imposing a direction indicator on the detected fiber direction is used for toolpath reconstruction, shown as Fig. 16. These imposed images are a good local toolpath representation of the tested sample in each sub-sectioned image. It clearly outlines the movement of the printing process, which can be applied to predict the hidden information in a 3D printed sample. Although, only 1 layer is used to demonstrate the toolpath reconstruction, the process can be repeated on the whole sample image stack to acquire the toolpath information of the entire sample. The result can be used as a blueprint for 3D printing reverse engineering or an 3D printing in-situ signature inspection.

## 4 Summary

The machine learning methods are now widely used in materials design. The opportunities presented by these methods have enabled design of materials with novel properties and reduced time to design complex composite materials for the requirements of specific applications. The present work shows the approaches that can be used for effectively processing the image datasets from materials with the example of a micro-CT scan image dataset processed by three different ML methods. A model composite material specimen is used and the ML approach is used to identify the toolpath used in 3D printing of this specimen. While these methods are useful for a variety of materials related problems such as design of new materials, processing parameter optimization and also defect detection in the microstructure, the ML methods also



**Fig. 16** **a** The to-be-test CT scan image showing the printing direction by glass fibers and **b** the collection of direction indicators with the trained model showing the toolpath of the certain layer

present challenges that they make the reverse engineering of the products easier. Although the size and geometry of a component can be 3D scanned very easily using available scanners and imaging tools, the quality of a component largely depends on the microstructure. The reverse engineering of microstructure by recovering the toolpath presents a vulnerability that can make reverse engineered products of high quality. The present work shows the need for developing new toolpath methodologies that are difficult to process through ML algorithms.

**Acknowledgements** National Science Foundation SaTC-EDU grant # 1931724 is acknowledged for supporting this work. The authors thank NYU Tandon School of Engineering Makerspace for the facilities provided for micro-CT scan.

## References

1. Kaschel FR, Vijayaraghavan RK, Shmeliov A, McCarthy EK, Canavan M, McNally PJ, Dowling DP, Nicolosi V, Celikin M (2020) Mechanism of stress relaxation and phase transformation in additively manufactured Ti-6Al-4V via in situ high temperature XRD and TEM analyses. *Acta Mater* 188:720–732
2. Spowart JE, Gupta N, Lehmhus D (2018) Additive manufacturing of composites and complex materials. *JOM* 70(3):272–274
3. Palmero EM, Casaleiz D, de Vicente J, Hernández-Vicen J, López-Vidal S, Ramiro E, Bollero A (2019) Composites based on metallic particles and tuned filling factor for 3D-printing by fused deposition modeling. *Compos Part A Appl Sci Manuf* 124:105497
4. Liu Z, Li M, Weng Y, Qian Y, Wong TN, Tan MJ (2020) Modelling and parameter optimization for filament deformation in 3D cementitious material printing using support vector machine. *Compos B Eng* 193:108018

5. Chen F, Mac G, Gupta N (2017) Security features embedded in computer aided design (CAD) solid models for additive manufacturing. *Mater Des* 128:182–194
6. Gupta N, Tiwari A, Bukkapatnam STS, Karri R (2020) Additive manufacturing cyber-physical system: supply chain cybersecurity and risks. *IEEE Access* 8:47322–47333
7. Ngo TD, Kashani A, Imbalzano G, Nguyen KTQ, Hui D (2018) Additive manufacturing (3D printing): a review of materials, methods, applications and challenges. *Compos B Eng* 143:172–196
8. Justo J, Távora L, García-Guzmán L, París F (2018) Characterization of 3D printed long fibre reinforced composites. *Compos Struct* 185:537–548
9. Averardi A, Cola C, Zeltmann SE, Gupta N (2020) Effect of particle size distribution on the packing of powder beds: a critical discussion relevant to additive manufacturing. *Mater Today Commun* 24:100964
10. Wang X, Jiang M, Zhou Z, Gou J, Hui D (2017) 3D printing of polymer matrix composites: a review and prospective. *Compos B Eng* 110:442–458
11. Heidari-Rarani M, Rafiee-Afarani M, Zahedi AM (2019) Mechanical characterization of FDM 3D printing of continuous carbon fiber reinforced PLA composites. *Compos B Eng* 175:107147
12. Zeltmann SE, Gupta N, Tsoutsos NG, Maniatakos M, Rajendran J, Karri R (2016) Manufacturing and security challenges in 3D printing. *JOM* 68(7):1872–1881
13. Yanamandra K, Chen GL, Xu X, Mac G, Gupta N (2020) Reverse engineering of additive manufactured composite part by toolpath reconstruction using imaging and machine learning. *Compos Sci Technol* 198:108318
14. Bartlett JL, Jarama A, Jones J, Li X (2020) Prediction of microstructural defects in additive manufacturing from powder bed quality using digital image correlation. *Mater Sci Eng A* 794:140002
15. Huang Z, Dantan J-Y, Etienne A, Rivette M, Bonnet N (2018) Geometrical deviation identification and prediction method for additive manufacturing. *Rapid Prototyping J* 24(9):1524–1538
16. Kyogoku H, Ikeshoji T-T (2020) A review of metal additive manufacturing technologies: mechanism of defects formation and simulation of melting and solidification phenomena in laser powder bed fusion process. *Mech Eng Rev* 7(1):19-00182–19-00182
17. Kim C, Espalin D, Cuaron A, Perez MA, MacDonald E, Wicker RB (2018) Unobtrusive in situ diagnostics of filament-fed material extrusion additive manufacturing. *IEEE Trans Compon Packag Manuf Technol* 8(8):1469–1476
18. Kuipers T, Doubrovski EL, Wu J, Wang CCL (2020) A framework for adaptive width control of dense contour-parallel toolpaths in fused deposition modeling. *Comput Aided Des* 128:102907
19. du Plessis A, Yadroitsava I, Yadroitsev I (2020) Effects of defects on mechanical properties in metal additive manufacturing: a review focusing on X-ray tomography insights. *Mater Des* 187:108385
20. Li W, Mac G, Tsoutsos NG, Gupta N, Karri R (2020) Computer aided design (CAD) model search and retrieval using frequency domain file conversion. *Addit Manuf* 36:101554
21. Comminal R, Serdeczny MP, Pedersen DB, Spangenberg J (2019) Motion planning and numerical simulation of material deposition at corners in extrusion additive manufacturing. *Addit Manuf* 29:100753
22. Honarvar F, Varvani-Farahani A (2020) A review of ultrasonic testing applications in additive manufacturing: defect evaluation, material characterization, and process control. *Ultrasonics* 108:106227
23. Deshpande AM, Minai AA, Kumar M (2020) One-shot recognition of manufacturing defects in steel surfaces. *Procedia Manuf* 48:1064–1071
24. Caggiano A, Zhang J, Alfieri V, Caiazzo F, Gao R, Teti R (2019) Machine learning-based image processing for on-line defect recognition in additive manufacturing. *CIRP Ann* 68(1):451–454
25. Chen F, Yu JH, Gupta N (2019) Obfuscation of embedded codes in additive manufactured components for product authentication. *Adv Eng Mater* 21(8):1900146
26. Chen F, Luo Y, Tsoutsos NG, Maniatakos M, Shahin K, Gupta N (2019) Embedding tracking codes in additive manufactured parts for product authentication. *Adv Eng Mater* 21(4):1800495

27. Chen F, Zabalza J, Murray P, Marshall S, Yu J, Gupta N (2020) Embedded product authentication codes in additive manufactured parts: imaging and image processing for improved scan ability. *Addit Manuf* 35:101319
28. Sacco C, Baz Radwan A, Anderson A, Harik R, Gregory E (2020) Machine learning in composites manufacturing: a case study of automated fiber placement inspection. *Compos Struct* 250:112514
29. Xu X, Gupta N (2019) Application of radial basis neural network to transform viscoelastic to elastic properties for materials with multiple thermal transitions. *J Mater Sci* 54(11):8401–8413
30. Xu X, Gupta N (2019) Artificial neural network approach to determine elastic modulus of carbon fiber-reinforced laminates. *JOM* 71(11):4015–4023
31. Meng L, McWilliams B, Jarosinski W, Park H-Y, Jung Y-G, Lee J, Zhang J (2020) Machine learning in additive manufacturing: a review. *JOM* 72(6):2363–2377
32. Liu Y, Guo B, Zou X, Li Y, Shi S (2020) Machine learning assisted materials design and discovery for rechargeable batteries. *Energ Storage Mater* 31:434–450
33. Khan A, Ko D-K, Lim SC, Kim HS (2019) Structural vibration-based classification and prediction of delamination in smart composite laminates using deep learning neural network. *Compos B Eng* 161:586–594
34. Dogan A, Birant D (2021) Machine learning and data mining in manufacturing. *Expert Syst Appl* 166:114060
35. Do DTT, Lee D, Lee J (2019) Material optimization of functionally graded plates using deep neural network and modified symbiotic organisms search for eigenvalue problems. *Compos B Eng* 159:300–326
36. Xu X, Gupta N (2019) Artificial neural network approach to predict the elastic modulus from dynamic mechanical analysis results. *Adv Theor Simul* 2(4):1800131
37. El Kadi H (2006) Modeling the mechanical behavior of fiber-reinforced polymeric composite materials using artificial neural networks—a review. *Compos Struct* 73(1):1–23
38. Sharma A, Kushvaha V (2020) Predictive modelling of fracture behaviour in silica-filled polymer composite subjected to impact with varying loading rates using artificial neural network. *Eng Fract Mech* 239:107328
39. Kwon O, Kim HG, Ham MJ, Kim W, Kim G-H, Cho J-H, Kim NI, Kim K (2020) A deep neural network for classification of melt-pool images in metal additive manufacturing. *J Intell Manuf* 31(2):375–386
40. Tian L, Fan Y, Li L, Mousseau N (2020) Identifying flow defects in amorphous alloys using machine learning outlier detection methods. *Scripta Mater* 186:185–189
41. Ko H, Witherell P, Lu Y, Kim S, Rosen DW (2020) Machine learning and knowledge graph based design rule construction for additive manufacturing. *Addit Manuf* 101620
42. Wang P, Fan E, Wang P (2020) Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. *Pattern Recogn Lett*. <https://doi.org/10.1016/j.patrec.2020.07.042>
43. Xu X, Gupta N (2018) Determining elastic modulus from dynamic mechanical analysis: a general model based on loss modulus data. *Materialia* 4:221–226
44. Xu X (2020) Machine learning approach to characterize elastic, viscoelastic, relaxation and creep behavior of materials. New York University Tandon School of Engineering, Ann Arbor, p 107
45. Xu X, Elgamal M, Doddamani M, Gupta N (2020) Tailoring composite materials for nonlinear viscoelastic properties using artificial neural networks. *J Compos Mater* 0021998320973744
46. Sabiston T, Inal K, Lee-Sullivan P (2020) Application of artificial neural networks to predict fibre orientation in long fibre compression moulded composite materials. *Compos Sci Technol*. <http://doi.org/10.1016/j.compscitech.2020.108034>
47. Yang J, Yao H (2020) Automated identification and characterization of two-dimensional materials via machine learning-based processing of optical microscope images. *Extreme Mech Lett* 39:100771
48. Kannala J, Rahtu E (2012) BSIF: binarized statistical image features. In: Proceedings of the 21st international conference on pattern recognition (ICPR2012). Tsukuba International Congress Center Tsukuba Science City, Japan

49. Raschka S, Mirjalili V (2017) Python machine learning. Packt Publishing Ltd., UK
50. Sak H, Senior AW, Beaufays F (2014) Long short-term memory recurrent neural network architectures for large scale acoustic modeling