INDUSTRIAL & SYSTEMS ENGINEERING
TEXAS A&M UNIVERSITY

ISEN 613 Engineering Data Analysis

Fall 2022

Final Project

Prediction of Additive Manufacturing Quality

Youssef Hebaish - 50%

Yinsu Zhang - 50%

December 14, 2022

Contents

List of Figures

List of Tables

1 Introduction

1.1 Importance of the Problem

Additive manufacturing is a newly involving manufacturing method that is highly customize and provides superior versatility and flexibility compared to the traditional methods of subtractive or formative [1]. The layer-by-layer construction also enables straightforward production of the design without the need of tooling development, saving time and money in the process; at the same time, adjustments or redesigns can be implemented easily with relatively low suck cost, and more importantly, speeding up the product development cycle for early product release from a business stand point [2]. In addition, the resulting components from additive manufacturing can be much more lightweight with lattice or hollow structure, achieving greater products on top of the previously mentioned reduction in manufacturing cost[2].

On the other hand, one of the biggest challenges in additive manufacturing remains to be surface defects like roughness, waviness, unwanted textures, ripples, voids, pores, scratches, etc [3]. The inconsistency of the additive manufacturing products become great risks when it comes to engineering applications like automotive and space vehicles, where the strength of the parts needs to meet certain standards consistently. Thus, it is crucial to be able to detect such defects during the manufacturing process especially because these surface defects would then be covered by other layers due to the natural of additive manufacturing.

1.2 Objective

The over arching objective of this project is to complete part of the qualifications of additive manufactured (3D printed) parts by detecting defects using sensor data collected during the printing process. Our goal is to develop machine learning methods that take sensor data as inputs in order to predict surface porosity features. Meanwhile, because additive manufacturing contains multiple stages like printing and milling in order to achieve optimal results, it is also important to study the effects of sensor signals from both printing and milling phases to determine what dataset would serve our models better.

1.3 Scope of Work

In this particular project, parts from a smart *OPTOMEC MTS 500 Hybrid Machine* were investigated for defects; specifically, the 1mm by 1mm surface images of the part captured by a high speed camera during the printing process were first processed to identify defects of scratches and pores. Apart from the high speed camera, the printer also captures continuous data throughout the printing process from accelerator, thin film piezoelectric sensor, dynamo-meter, FlexiForce sensor, Marposs CMM, acoustic emission sensor, power sensor, temperature sensor, and STIL Chromline.

For the purposes of this project, we were provided with the following: 1mm by 1mm images of the part surface which contain possible defects; an excel spreadsheet containing extracted porosity information from exsisting image processing efforts (this can be used as ground truth for the prediction models); a MATLAB data file with acoustic emission and acceleration data collected during both the printing process and the milling process (this will be the input to our prediction models). In the following sections, we will investigate both regression and classification models and try to predict the *% area porosity* and *pore count* of individual 1mm by 1mm surfaces of the part. In addition, models will be build using both printing and milling data, only printing data, or only milling data to determine what features would be ideal predictors for *% area porosity* and/or *pore count* output.

2 Project Approach

In this project, we form two main classes of problems: regression and classification. While classification is the more common approach in quality control settings in terms of predicting whether a part is defective or not, regression could be useful in providing insights into different process settings and their effect on quality. Therefore, we experiment with regression as well as classification. The data used in this project is the provided ***sample 1***. There are 72 data points for which the response is available in the provided ground truth file. Description of the used data is discussed further in the coming subsections.

In this section, we discuss the proposed project approach and the methodology used for each approach. We, first, discuss the regression models in terms of what predictors are used and the predicted response in Section 2.1.1. We also discuss the methodology used in building those models in Section 2.1.2. Then, similarly, we discuss the classification approach taken to build prediction models in Section 2.2.1 and the methodology used in building those models in Section 2.2.1.

2.1 Regression

Regression is a statistical technique used in quality control to measure the relationship between two or more variables [4]. It is used to identify the extent to which changes in one variable are associated with changes in another. For example, in manufacturing, regression can be used to identify the influence of changes in production process parameters on the quality of the final product [5].

Regression can also be used to identify and quantify the influence of other factors on quality. For example, in a customer satisfaction survey, regression can be used to identify the influence of various customer demographic and psychographic characteristics on customer satisfaction [4].

Regression can be used to identify the optimal conditions for a given process parameter or set of parameters. This can be useful in quality control to identify the best combination of process parameters to produce the desired quality level [5]. Additionally, regression can be used to identify any potential sources of variability in a process. This can be helpful in identifying potential causes of non-conformance and taking corrective actions. In this subsection, we discuss the different regression models we built as well as the methodology used in building those models.

2.1.1 Regression Models

We build three regression models that share the same response, the percentage area porosity, but differ in the used predictors: (i) Combined printing and milling channels, (ii) Printing only, and (iii) Milling only. The purpose of the first model, illustrated in Table 1, is to test the predictability of the entire process on the quality of the part. The purpose of the second and the third models, illustrated in Tables 2 and 3, is to isolate each processes, printing and milling, and see how each one affect the percentage area porosity. That is, we explore whether printing or milling is more closely related to the response. There are four printing channels and two milling channels. We utilize ***sample 1***, which consists of 72 data points. We use cross validation, which is discussed in detail in Section 3.

Table 1: Data description of the combined regression model.

#	**Predictors**	**Response**
1	$(6 \times 15 \times 129)$	%Area Porosity
...	...	...
72	$(6 \times 15 \times 129)$	%Area Porosity

Table 2: Data description of the printing regression model.

#	**Predictors**	**Response**
1	$(4 \times 15 \times 129)$	%Area Porosity
...	...	...
72	$(4 \times 15 \times 129)$	%Area Porosity

Table 3: Data description of the milling regression model.

#	**Predictors**	**Response**
1	$(2 \times 15 \times 129)$	%Area Porosity
...	...	...
72	$(2 \times 15 \times 129)$	%Area Porosity

As shown in Tables 1-3, predictors are not one dimensional. In fact, each data point is three-dimensional, which complicates the analysis a little bit. That is, conventional regression methods such as linear regression or multiple linear regression would not be suitable for this kind of input dimensions. For instance, in order to use linear regression, multiple linear regression has to be used and each data point has to be flattened. However, if each data point is flattened, that would result in a significantly large number of predictors, which is neither reasonable nor practical. Therefore, we resort to deep learning methods, which is capable of dealing with multidimensional inputs. We discuss the method used for the three models in Section 2.1.2. The used data is the 72 data points in ***sample 1***.

2.1.2 Regression Methodology

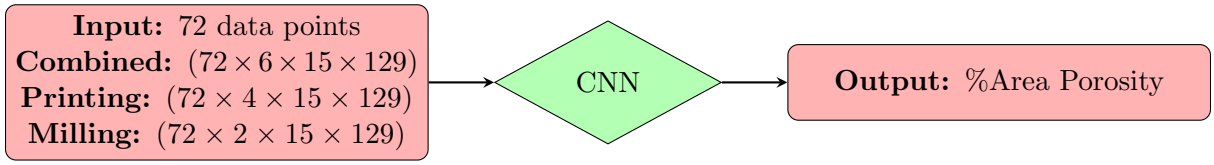
Deep learning is a branch of artificial intelligence (AI) that is based on the use of neural networks to perform complex tasks such as image recognition, natural language processing, and decision-making [6]. Deep learning algorithms are composed of multiple layers of neurons, which are connected in a manner similar to how the neurons in the human brain are connected. Each neuron in a layer receives input from the neurons in the preceding layer and generates an output, which is then passed to the neurons in the succeeding layer. The layers in a deep learning network are typically organized in a hierarchical fashion, with each layer learning increasingly abstract features of the data.

Deep learning algorithms have been applied to a wide range of tasks, such as image classification, object detection, natural language processing, and autonomous driving [7]. Deep learning algorithms are well-suited for these tasks because they can learn from large and complex datasets, recognize patterns in data, and make predictions with high accuracy. Additionally, deep learning algorithms can be trained on GPUs, allowing for fast and efficient training.

Convolutional Neural Networks (CNNs) have become a popular tool in the field of deep learning for tasks such as object detection, image classification, and segmentation [8]. CNNs are a type of artificial neural network (ANN) that is specifically designed for processing 2-dimensional inputs, such as images. CNNs are composed of several layers, including the input layer, convolutional layers, pooling layers, and a fully-connected output layer. The convolutional layers are responsible for extracting features from the input image, while the pooling layers reduce the resolution of the extracted features to make them easier to process. The output layer uses the extracted features to generate a prediction.

CNNs have several advantages over traditional ANNs, such as their ability to recognize patterns in data, their ability to learn from smaller datasets, and their ability to process large amounts of data quickly [6]. Additionally, CNNs are well-suited for tasks where the input data is highly structured, such as images. For example, CNNs have been used to classify objects in images, detect faces, and segment objects in videos [9]. Therefore we use CNNs in all our 3 regression models to predict the percentage area porosity of each voxel. The architecture of the models are discussed in detail in Section 3. Figure 1 shows the flowchart of the discussed regression models.

Figure 1: Regression models schematic



2.2 Classification

Classification in machine learning is an important tool in quality control. Classification algorithms are used to assign quality labels to different products or services. For example, in manufacturing, a classification algorithm can be used to assign a quality label to a product based on its features and characteristics [10]. By using a classification algorithm, manufacturers can ensure that their products meet the required quality standards.

In addition to quality control, classification algorithms are also used for other tasks such as object recognition and image processing [11]. In object recognition, a classification algorithm is used to distinguish between different objects based on their shapes and sizes. In image processing, classification algorithms are used to identify different patterns in images.

Classification algorithms are also used in customer service. A classification algorithm can be used to classify customer feedback and provide an accurate response. For example, customer feedback can be classified as positive, negative, or neutral. Based on the classification, the customer service representative can then provide a more appropriate response [12]. In this subsection, we discuss the different regression models we built as well as the methodology used in building those models.

2.2.1 Classification Models

The approach used in classification is very similar to that used in regression. We construct 6 different models that vary in response as well as predictors. That is, the 6 models are split into two sets of three models that are the same response. All 6 models are summarized in Table 4

Table 4: Classification models.

Model #	**Predictors**	**Response**
1	$(6 \times 15 \times 129)$	%Area Porosity
2	$(4 \times 15 \times 129)$	%Area Porosity
3	$(2 \times 15 \times 129)$	%Area Porosity
4	$(6 \times 15 \times 129)$	Pore Count
5	$(4 \times 15 \times 129)$	Pore Count
6	$(2 \times 15 \times 129)$	Pore Count

Percentage area porosity and the pore count is converted to a binary classification output to be used in classification. Percentage area porosity is set to 1, which is 'bad', if the value is above the mean of all values, and it is set to 0, which is 'good', if the value is below the mean of all values. Similarly, for pore count, the value is set to 1 if it is above the median of all values and 0 if it is below. The purpose of those 6 models is to explore the effect of manufacturing settings on different quality measures. The two quality measures are the percentage area porosity and the pore count, which are closely related but might be affected differently by different settings. We also explore of printing or milling channels have an effect on a particular quality measure more than the other. We run into the same difficulty of multidimensional inputs we faced in regression, and hence, we resort to deep learning techniques, which are discussed next.

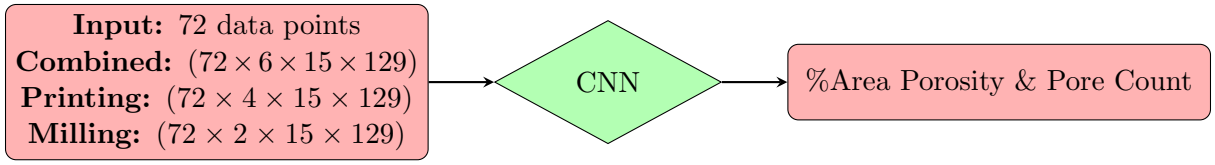
2.2.2 Classification Methodology

Deep learning is a type of artificial intelligence that uses neural networks to learn from large amounts of data. It is a subset of machine learning and it has become increasingly popular due to its ability to produce accurate results in a variety of areas. Classification is one of the most common applications of deep learning, and it involves using algorithms to classify data into different categories. Classification in deep learning usually involves using artificial neural networks. Neural networks are composed of layers of interconnected neurons and each neuron acts as a filter for the input data. By adjusting the parameters of the neurons, the network can learn to classify data into different categories. This process is known as supervised learning, where the network is trained on a labeled dataset. After training, the network can then be used to classify new data.

In classification tasks, deep learning algorithms are usually used to classify images or text. For example, a deep learning algorithm can be used to classify images of cats and dogs into the appropriate categories. Similarly, a deep learning algorithm can be used to classify text into different categories, such as positive or negative sentiment. In addition to classification, deep learning can also be used for other tasks such as object detection, image segmentation, and natural language processing. Deep learning can also be used for recommendation systems, where it can be used to recommend items to users based on their previous interactions.

In recent years, convolutional neural networks (CNNs) have become increasingly popular in the field of quality control. CNNs are a type of artificial neural network used for image recognition. They are composed of several layers of neurons, where each neuron is connected to a specific region of an input image, allowing the network to learn patterns from the data. By understanding the patterns in the data, a CNN can be used to detect defects in products or images, allowing for automated quality control. CNNs have been used for various quality control tasks, such as defect detection in textiles and semiconductors . In these applications, CNNs are used to learn the features of a product or image that indicate a defect.In addition to defect detection, CNNs have also been used for automated quality control of images. Therefore, we use CNNs in constructed our models, which is discussed in Section 3. Similar to regression models, the flowchart of classification models are shown in Figure 2.

Figure 2: Classification models schematic



3 Implementation

In this section, we discuss the architecture of each regression and classification model. We also discuss and interpret the results of each model. First, we discuss how regression models are implemented and their results in Section 3.1. Them we discuss classification models and they are constructed in Section 3.2. However, there is a common step taken for both types of model, which is input processing. Inputs are presented in *.mat* files, where each file represents a data point of dimensions $6 \times 15 \times 129$. All models are created from ***sample 1***, having a total of 72 data points. the mat files are loaded and stored in 3 different variables. The first variable, which has a shape $72 \times 6 \times 15 \times 129$, is one that has the combined printing and milling channels, which has the dimension $6 \times 15 \times 129$. The second variable, which has a shape $72 \times 4 \times 15 \times 129$, stores inputs from printing channels only, with each data point of dimension $4 \times 15 \times 129$. The third variable, which has a shape $72 \times 2 \times 15 \times 129$, stores data points with milling channels only.

In addition to input processing, output, or response, mapping is done by loading data from the ground truth excel file. In particular, 3 columns are of interest: Voxel Index, Pore Count, %Area Porosity. Each input data point is paired with its output using Voxel Index. This step was necessary because, later on, data points will be shuffled, and therefore, it is important to map responses to input data points. Those previous two steps are crucial in the architecture of all regression and classification models, and hence, they are universal to all models.

3.1 Regression

As previously mentioned, we construct 3 regression models with the response being the %Area Porosity the inputs are the 72 data points in ***sample 1***. We use CNNs for the 3 models. The architecture of the three models is identical with the only difference being in input shape as previously mentioned. The architecture of the CNN used is as follows:

- We learn a total of 6 filters
- A Max Pool is used to reduce the spatial dimensions
- We then learn 1 filters
- A Max Pool is used
- We flatten the input
- We add a dense layer with 6 neurons
- We add a dense layer with 1 neuron

For the three models, input data is split into training and testing sets. 64 data points are used in training, while 8 points are using for testing. For the combined model, 200 epochs are used in training, with MSE being the accuracy measure. Then, testing is done using the 8 data points, with MSE as the accuracy measure. For printing and milling models, 500 epochs are used. Indeed, data points are shuffled after loading before training the model. Results of the three models are shown in Table 5. One of the difficulties of regression models is assessing it accuracy. MSE, which is a measure of accuracy or deviation, is not an absolute indicator of accuracy. An ideal value of MSE is as close as much to zero, but that is unrealistic. Therefore, it is difficult to judge the model's accuracy based on MSE value. However, MSE is sensitive to the scale of the data. That is, MSE might be large because the magnitude of the predicted values are large, and vice versa. In our case, this value of MSE is relatively high because values of the predicted response are not high.

However, these results provide important insights as to the comparison between printing and milling. We notice that the value of MSE is the same for the combined model and printing model, which means that printing channels are influential in predicting the %Area Porosity. Milling MSE is higher than those of the combined and the printing models, which means that milling channels are not as influential as printing channels. Nonetheless, we have to say that the size of the input data is not sufficiently large to produce reliably accurate results.

Table 5: Results of regression models.

#	**Model**	**MSE**
1	Combined	1.77
2	Printing	1.77
3	Milling	1.83

3.2 Classification

We construct 2 sets of classification models, each has 3 models as shown in Table 4. The first step, however, which is a universal step to all models, is to convert the response to a binary classification response. This is done to two responses: Pore Count and %Area Porosity. For Pore Count, the response is 0, which is 'good', if the value is below the median (5.5); and 1, which is 'bad', if it is above the median. For %Area Porosity, the response is 0, which is 'good', if the value is below the mean (1.66) and 1, which is 'bad', if it is above the mean. Classification models are training and cross validated using k-fold cross validation with 8 folds each of size 8. We, then, test the model using the test dataset (8 points) by using the models from each fold. The input, CNN architecture, and output of all 6 models are summarized in Table 6. In testing the models, we plot the accuracy of each fold in each model.

Figures 3 and 4 show the results of the two sets of classifications models in predicting Pore Count and %Area Porosity. Figure 3 show the accuracy of the three classification models in predicting Pore count. We observe that the combined model has an average accuracy across folds that is higher than the two separate models. Nonetheless, we also observe that the maximum accuracy of the printing model is higher than that of the milling model. This is observable by looking at folds 1 and 6 of the printing model, which have an accuracy of approximately 88%, while the maximum accuracy of the milling model is 75% at fold 4. However, on the other hand, the minimum accuracy of the printing model is less than that of the milling model. Therefore, it is very difficult to determine with great certainty the model with the higher accuracy in general. The main reason for the fluctuation in accuracy between folds is that the testing, as well as the training, dataset is small (only 8 points). We think models accuracy might converge if the test dataset is larger, which is usually the case in general.

Figure 4 show the accuracy of the three models in predicting %Area Porosity. Similarly, the average accuracy of the printing model is higher than those of the combined and the milling models. However, the maximum accuracy of the three models is, approximately, equal at 75%. The minimum accuracy is equal among the three models as well. The issue of the size of the testing dataset remains the main reason for the fluctuation in accuracy. We compare the accuracy of different models and approaches in Section 4.

Table 6: Classification models description and construction.

#	**Input & Shape**	**CNN Architecture**	**Output**
1	Combined dataset ($6 \times 15 \times 129$)	• Layer with 6 filters • Max Pool • Layer with 1 filter • Max Pool • Flatten inputs • Dense layer with 6 neurons • Dense layer with 1 neuron	Pore Count
2	Printing dataset ($4 \times 15 \times 129$)	• Layer with 6 filters • Max Pool • Layer with 1 filter • Max Pool • Flatten inputs • Dense layer with 6 neurons • Dense layer with 1 neuron	Pore Count
3	Milling dataset ($4 \times 15 \times 129$)	• Layer with 6 filters • Max Pool • Layer with 1 filter • Flatten inputs • Dense layer with 6 neurons • Dense layer with 1 neuron	Pore Count
4	Combined dataset ($6 \times 15 \times 129$)	• Layer with 6 filters • Max Pool • Layer with 1 filter • Max Pool • Flatten inputs • Dense layer with 6 neurons • Dense layer with 1 neuron	%Area Porosity
5	Printing dataset ($4 \times 15 \times 129$)	• Layer with 6 filters • Max Pool • Layer with 1 filter • Max Pool • Flatten inputs • Dense layer with 6 neurons • Dense layer with 1 neuron	%Area Porosity
6	Milling dataset ($4 \times 15 \times 129$)	• Layer with 6 filters • Max Pool • Layer with 1 filter • Flatten inputs • Dense layer with 6 neurons • Dense layer with 1 neuron	%Area Porosity

Figure 3: Classification models accuracy in predicting Pore Count.

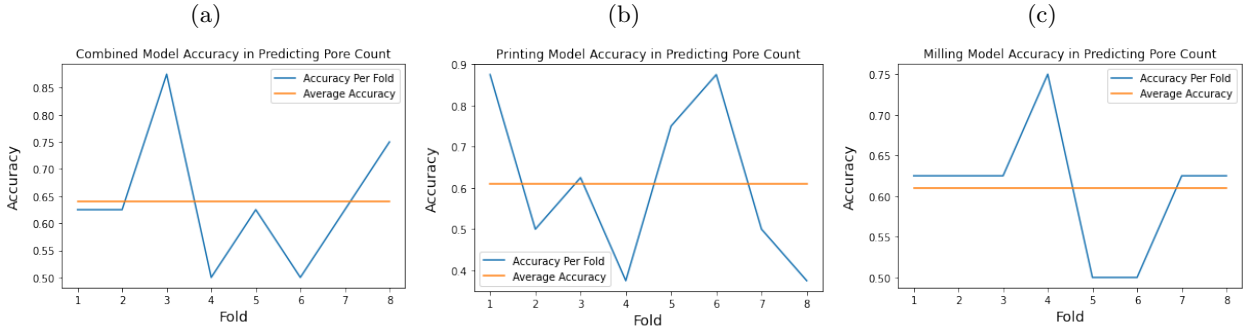
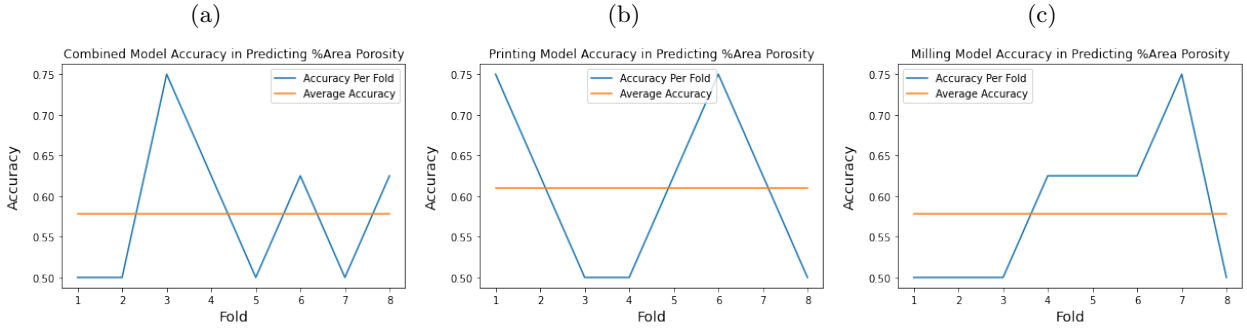


Figure 4: Classification models accuracy in predicting % Area Porosity.



4 Comparison

As shown previously in Section 2.1.2, CNN was our chosen method for regression to predict *% area porosity* because it performs well on well structured dataset like the one provided to us during this project. According to the results of the 3 CNN models presented in Table 5, compared to the milling data set. the combined dataset of both milling and printing yielded the same low MSE of 1.77, suggesting that printing maybe an overall larger contributing factor to the *% Area Porosity* that was being predicted.

Meanwhile, CNN was also utilized for our classification models to predict both *% Area Porosity* and *Pore Count* with the detailed architecture construction shown in Table 6. As found previously in Section 3.2, although whether milling data alone is a a good predict or not in predicting *Pore Count* during our classification models remains inconclusive, the combined model resulted in the highest average prediction accuracy as shown in Figure 3. In addition, the same figure showed the highest prediction accuracy was predicted by the printing data alone.

Overall, we believe the printing data may be the more contributing predictor for both the regression and classification models because pores are usually created within the printing stage of the process; while milling data may predict the *% Area Porosity* better since the milling process theoretically enlarges the pores when it tries to flatten the surface, exposing more of the pore structure to the surface.

5 Executive Summary

This course project aims to provide a novel method of identifying common porous defects during the additive manufacturing process because there is a growing need of qualification standards for metal additive manufacturing [3]. Defects within layers of the printed parts are be detrimental to its final quality, so the ability to detect such defect quickly and accurately is crucial. One of the widely used pore detection methods is by using x-ray tomography [13], which is both costly and time consuming. In this project, we explored the possibility of using sensor outputs from a smart machine to predict porous defects within small vortexes of the surface. This method is much more cost effective and faster compared to the traditional x-ray tomography, and it has the potential to detect defects during the manufacturing process instead of after completion. Our results showed that the regression model of predicting *% Area Porosity* can provide useful insights on whether the sensor data from the printing or the milling stage contribute differently to the porous prediction results. In addition, the classification models of using milling data only, printing data only, and combined data as predictors to predict both *Pore Count* and *% Area Porosity* suggests the prediction accuracy may have not reached sufficient level (75%) for such model to be deployable to real-world usages, but the model prediction accuracy is not far off (65%). Our findings suggests machine learning models using smart machine sensor data to predict surface defects have great potential to be a reliable tool for additive manufacturing quality control.

6 Conclusions

Our best regression model returned MSE value of 1.77, which can provide us insights about the quality of the individual vortexes, but it cannot determine whether a part is up to the standard or not. The classification models with defined thresholds gives the ability to actually classify a vortex to be good or bad. However, the fact that none of our classification models achieved the "acceptable prediction accuracy above 70%" mentioned in class shows us that our classification models are probably not deploy-able to determine weather a part is up to the standard or not. It is possible that a larger dataset can yield us more accurate models. The results also reminds me of the initial example code provided to us defined the threshold as: good if *% Area Porosity* is less than 1%; and bad if *% Area Porosity* is above 2%. This method would be able to return better prediction accuracy; however this model may leave a lot of vortexes to be in the "grey area" and require further qualification efforts to classify.

Future work can be done in exploring different machine learning methods to better predict defects in the printing process, especially when qualifications of additive manufacturing remains one of the major challenges before such manufacturing methods can be made more popular. Potential real time predicting capabilities are also worth exploring, because if the part in the making can be determined to be seriously defective, the job can be aborted in order to either savage the part before additional layers are put on, or scrap it to save materials and machine time during the process.

References

[1] Syed AM Tofail, Elias P Koumoulos, Amit Bandyopadhyay, Susmita Bose, Lisa O'Donoghue, and Costas Charitidis. Additive manufacturing: scientific and technological challenges, market uptake and opportunities. *Materials today*, 21(1):22–37, 2018.

[2] Zhenzhen Quan, Amanda Wu, Michael Keefe, Xiaohong Qin, Jianyong Yu, Jonghwan Suhr, Joon-Hyung Byun, Byung-Sun Kim, and Tsu-Wei Chou. Additive manufacturing of multi-directional preforms for composites: opportunities and challenges. *Materials Today*, 18(9): 503–512, 2015.

[3] Anton Du Plessis, Ina Yadroitsava, and Igor Yadroitsev. Effects of defects on mechanical properties in metal additive manufacturing: A review focusing on x-ray tomography insights. *Materials & Design*, 187:108385, 2020.

[4] Michael A Golberg and Hokwon A Cho. *Introduction to regression analysis*. WIT press, 2004.

[5] Amitava Mitra. *Fundamentals of quality control and improvement*. John Wiley & Sons, 2016.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[10] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[11] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[12] Pat Langley, Wayne Iba, Kevin Thompson, et al. An analysis of bayesian classifiers. In *Aaai*, volume 90, pages 223–228, 1992.

[13] Anton Du Plessis, Stephan G le Roux, Jess Waller, Philip Sperling, Nils Achilles, Andre Beerlink, Jean-François Métayer, Mirko Sinico, Gabriel Probst, Wim Dewulf, et al. Laboratory x-ray tomography for metal additive manufacturing: Round robin test. *Additive Manufacturing*, 30:100837, 2019.

Appendix

A Code

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
!ls "/content/drive/Shareddrives/ISEN 613 Final
Project/ISEN613-Engineering-Data-Analysis-CourseProject-main"

# cnn model and accuracy - porosity tensor dataset
# Import packages / libraries

import os
from os.path import dirname, join as pjoin
import scipy.io
import tensorflow as tf
import torch

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

from os import makedirs
from os import listdir
from shutil import copyfile
from random import seed
from random import random

# baseline model for the classification dataset
import sys
from matplotlib import pyplot
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from sklearn.model_selection import KFold, StratifiedKFold
from tensorflow.keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator

!pip install opencv-python
!pip install lime
!pip install lazypredict

import numpy as np
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
```

```
from skimage.color import gray2rgb
from skimage.color import rgb2gray
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import inception_v3 as inc_net
from lime import lime_image
import pandas as pd
import glob
from tensorflow.keras.models import load_model
from sklearn.metrics import r2_score
import cv2
from keras.preprocessing.image import ImageDataGenerator
from skimage.segmentation import mark_boundaries

## LIME ANALYSIS
## Import required libraries / packages

import numpy as np
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
import math
import scipy
import scipy.io
from PIL import Image
from scipy import ndimage
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
# import tensorflow_addons as tfa
import pydot
import pydotplus
import graphviz
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import Model
import random
from keras.models import load_model
import pickle
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import glob
import os
import pandas as pd

import types
```

```
from lime.utils.generic_utils import has_arg
from skimage.segmentation import felzenszwalb, slic, quickshift
import copy
from functools import partial

import sklearn
import sklearn.preprocessing
from sklearn.utils import check_random_state
from skimage.color import gray2rgb
from tqdm.auto import tqdm

import scipy.ndimage as ndi
from skimage.segmentation._quickshift_cy import _quickshift_cython

from lime import lime_base
from lime.wrappers.scikit_image import SegmentationAlgorithm

import skimage
from matplotlib import colors
from skimage.segmentation import mark_boundaries, find_boundaries
from skimage.morphology import dilation,square
from collections import Counter
import re
from sklearn.ensemble import RandomForestRegressor

from numpy.core.fromnumeric import shape
# organize dataset into a numpy structure
# Extract input spectrogram data from .mat files
# Loading the ground truth xlsx file
gt = pd.read_excel(r"/content/drive/Shareddrives/ISEN 613 Final Project
    /ISEN613-Engineering-Data-Analysis-CourseProject-main/CNN code and
    input/Ground Truth - Voxel Tensor Index - Sample1.xlsx")
perc_area = gt.iloc[0:90, [1,5]]
pore_count = gt.iloc[0:90,[1,2]]

np.set_printoptions(formatter={'float': lambda x:
"{0:0.3f}".format(x)})
# sets print options to 5 decimal places

TENSOR_INPUT_DATA_COMBINED = torch.ones((72,6,129,15))
TENSOR_INPUT_DATA_PRINTING = torch.ones((72,4,129,15))
TENSOR_INPUT_DATA_MILLING = torch.ones((72,2,129,15))

    # 72 - Total number of data points for binary classification
    # 6 - Channels (pertaining to each spectrogram)
    # 129  - Freuqency bands for each spectrogram
    # 15 - Time steps
```

```
voxel_number = 0

# assign directory
# directory = 'files'
# iterate over files in that directory
# for filename in os.listdir(directory):


directory = (r"/content/drive/Shareddrives/ISEN_613_Final
Project/ISEN613-Engineering-Data-Analysis-CourseProject-main/
CNN_code_and_input/Tensor_data_72_voxels_sample1")
# Change this directory location to your folder that contains
the 72 .mat files from Tensor_data_72_voxels.zip corresponding
to Sample 1.
# When working with Sample 4 (Tensor_data_90_voxels_sample4.zip),
extract the 90 files in a folder, and update the directory.
In addition,
# update the dimensions in the below for loop (and in other places
as required) as each tensor in Sample 4 is of dimension 129x6x4,
# ie, 129 frequency bands, 6 time steps, 4 spectrograms pertaining to
only printing cycle.

voxel_perc_area = []
voxel_pore_count = []
class_comb_area = np.zeros((72,2))
class_comb_count = np.zeros((72,2))
reg_comb = np.zeros((72,2))

for filename in os.listdir(directory):
    # print(filename)
    mat_fname = pjoin(directory, filename)
    Tensor_data_voxel = scipy.io.loadmat(mat_fname)

    data = list(Tensor_data_voxel.items())
    tensor_array = np.array(data, dtype=object)

    # The spectrogram data is captured in a list from one of
    the elements of tensor_array
    # Shape of tensor_array is (4,2) and all spectrogram data (129x15x6
        )
    is stored in tensor_array[3][1]

    spec_list = list(tensor_array[3][1])

    # Extracting the desired output to build the different datasets
    voxel_name = tensor_array[3][0]
    voxel_ind = [int(s) for s in re.findall(r'\d+', voxel_name)][0]
    voxel_perc_area.append(np.array(perc_area.loc[perc_area['
```

```
        Tensor_voxel__index']
    == voxel_ind, '%Area_Porosity'])[0])
    voxel_pore_count.append(np.array(pore_count.loc[perc_area['
        Tensor_voxel__index']
    == voxel_ind, 'Pore_Count'])[0])

    for i in range(len(spec_list)):
        for j in range(15):
            for k in range(6):
                TENSOR_INPUT_DATA_COMBINED[voxel_number,k,i,j] =
                    spec_list[i][j][k]

    TENSOR_INPUT_DATA_PRINTING = TENSOR_INPUT_DATA_COMBINED[:,(1,2,4,5)
        ,:,:]
    TENSOR_INPUT_DATA_MILLING = TENSOR_INPUT_DATA_COMBINED[:,(0,3),:,:]

    voxel_number = voxel_number + 1

# for i in range(72):
#   class_comb_area[i][0] = tensor_np[i]
#   class_comb_count[i][0] = tensor_np[i]
#   reg_comb[i][0] = tensor_np[i]

#   class_comb_area[i][1] = voxel_perc_area[i]
#   class_comb_count[i][1] = voxel_pore_count[i]
#   reg_comb[i][1] = voxel_perc_area[i]


y_target_area = np.array([voxel_perc_area[i] for i in range(72)])
y_target_count = np.array([voxel_pore_count[i] for i in range(72)])

# y_target[4:10] = 1
# y_target[13:18] = 1
# y_target[22:34] = 1
# y_target[37:41] = 1
# y_target[43:46] = 1
# y_target[47] = 1
# y_target[57] = 1
# y_target[59] = 1
# y_target[70] = 1

data_order = np.arange(72)
np.random.shuffle(data_order)

'''
print("----------------------")
print(TENSOR_INPUT_DATA_COMBINED.shape)
print(TENSOR_INPUT_DATA_COMBINED)
```

```
print(TENSOR_INPUT_DATA_PRINTING.shape)
print(TENSOR_INPUT_DATA_PRINTING)
print(TENSOR_INPUT_DATA_MILLING.shape)
print(TENSOR_INPUT_DATA_MILLING)
print("----------------------")
print(y_target)
print("----------------------")
data_order = np.arange(72)
np.random.shuffle(data_order)
print(data_order)
'''

# Saving data as .npy files
NUMPY_INPUT_DATA = TENSOR_INPUT_DATA_COMBINED.numpy()
NUMPY_INPUT_DATA_printing = TENSOR_INPUT_DATA_PRINTING.numpy()
NUMPY_INPUT_DATA_milling = TENSOR_INPUT_DATA_MILLING.numpy()

np.save(r"/content/drive/Shareddrives/ISEN 613 Final Project/ISEN613-
    Engineering-Data-Analysis-CourseProject-main/CNN code and input/
    Saved Inputs_Outputs/Comb_Inputs", NUMPY_INPUT_DATA)
np.save(r"/content/drive/Shareddrives/ISEN 613 Final Project/ISEN613-
    Engineering-Data-Analysis-CourseProject-main/CNN code and input/
    Saved Inputs_Outputs/Print_Inputs", NUMPY_INPUT_DATA_printing)
np.save(r"/content/drive/Shareddrives/ISEN 613 Final Project/ISEN613-
    Engineering-Data-Analysis-CourseProject-main/CNN code and input/
    Saved Inputs_Outputs/Mill_Inputs", NUMPY_INPUT_DATA_milling)
np.save(r"/content/drive/Shareddrives/ISEN 613 Final Project/ISEN613-
    Engineering-Data-Analysis-CourseProject-main/CNN code and input/
    Saved Inputs_Outputs/Target_Area", y_target_area)
np.save(r"/content/drive/Shareddrives/ISEN 613 Final Project/ISEN613-
    Engineering-Data-Analysis-CourseProject-main/CNN code and input/
    Saved Inputs_Outputs/Target_Count", y_target_count)

NUMPY_INPUT_DATA = TENSOR_INPUT_DATA_COMBINED.numpy()

X_train_reg = NUMPY_INPUT_DATA[data_order[0:64],:,:,:]
y_train_reg = y_target_area[data_order[0:64]]
X_test_reg = NUMPY_INPUT_DATA[data_order[64:72],:,:,:]
y_test_reg = y_target_area[data_order[64:72]]

# Define the model architecture
model = Sequential()
model.add(Conv2D(6, (7, 7), activation='relu', kernel_initializer='
    he_uniform', padding='same', input_shape=(6, 129, 15)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(1, (7, 7), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
```

```
# model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
# model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(6, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
#opt = SGD(lr=0.001, momentum=0.9)

model.compile(loss='mean_squared_error', optimizer='adam', metrics='mse
    ')

model.fit(X_train_reg, y_train_reg,
          epochs=200,
          verbose=1,
          validation_data=(X_test_reg, y_test_reg))
score = model.evaluate(X_test_reg, y_test_reg, verbose=0)
print('Model_MSE:', score[0])

NUMPY_INPUT_DATA_printing = TENSOR_INPUT_DATA_PRINTING.numpy()

X_train_reg_printing = NUMPY_INPUT_DATA_printing[data_order
    [0:64],:,:,:]
y_train_reg = y_target_area[data_order[0:64]]
X_test_reg_printing = NUMPY_INPUT_DATA_printing[data_order
    [64:72],:,:,:]
y_test_reg = y_target_area[data_order[64:72]]

# Define the model architecture
model = Sequential()
model.add(Conv2D(6, (7, 7), activation='relu', kernel_initializer='
    he_uniform', padding='same', input_shape=(4, 129, 15)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(1, (7, 7), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
# model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
# model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(6, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
#opt = SGD(lr=0.001, momentum=0.9)

model.compile(loss='mean_squared_error', optimizer='adam', metrics='mse
```

```
    ')

model.fit(X_train_reg_printing, y_train_reg,
          epochs=500,
          verbose=1,
          validation_data=(X_test_reg_printing, y_test_reg))
score = model.evaluate(X_test_reg_printing, y_test_reg, verbose=0)
print('Model MSE:', score[0])


NUMPY_INPUT_DATA_milling = TENSOR_INPUT_DATA_MILLING.numpy()

X_train_reg_milling = NUMPY_INPUT_DATA_milling[data_order[0:64],:,:,:]
y_train_reg = y_target_area[data_order[0:64]]
X_test_reg_milling = NUMPY_INPUT_DATA_milling[data_order[64:72],:,:,:]
y_test_reg = y_target_area[data_order[64:72]]

# Define the model architecture
model = Sequential()
model.add(Conv2D(6, (7, 7), activation='relu', kernel_initializer='
    he_uniform', padding='same', input_shape=(2, 129, 15)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(1, (7, 7), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
#model.add(MaxPooling2D((2, 2)))
# model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
# model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(6, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
#opt = SGD(lr=0.001, momentum=0.9)

model.compile(loss='mean_squared_error', optimizer='adam', metrics='mse
    ')

model.fit(X_train_reg_milling, y_train_reg,
          epochs=500,
          verbose=1,
          validation_data=(X_test_reg_milling, y_test_reg))
score = model.evaluate(X_test_reg_milling, y_test_reg, verbose=0)
print('Model MSE:', score[0])


# Specifying the response and converting it to 0,1 type
y_class_count = np.zeros(72)
```

```
for i in range(72):
  if y_target_count[i]<=np.median(y_target_count):
    y_class_count[i] = 0
  else:
    y_class_count[i] = 1


# CNN Model Architecture - Printing and Milling combined cycles

NUMPY_INPUT_DATA = TENSOR_INPUT_DATA_COMBINED.numpy()


X_train_class = NUMPY_INPUT_DATA[data_order[0:64],:,:,:]
y_train_class_count = y_class_count[data_order[0:64]]
X_test_class = NUMPY_INPUT_DATA[data_order[64:72],:,:,:]
y_test_class_count = y_class_count[data_order[64:72]]


# Define the K-fold Cross Validator
kfold = KFold(n_splits=8, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 1
train_acc_per_fold = []
train_loss_per_fold = []
val_acc_per_fold = []
val_loss_per_fold = []
test_acc_per_fold = []
test_loss_per_fold = []

for train, val in kfold.split(X_train_class, y_train_class_count):

    train_acc = 0
    val_acc = 0
    test_acc = 0
    # Define the model architecture
    while(train_acc < 0.5 or val_acc < 0.5 or test_acc < 0.5):
        model = Sequential()
        model.add(Conv2D(6, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same', input_shape
            =(6, 129, 15)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(1, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        # model.add(Conv2D(128, (3, 3), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        # model.add(MaxPooling2D((2, 2)))
```

```
    model.add(Flatten())
    model.add(Dense(6, activation='relu', kernel_initializer='
        he_uniform'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(loss='binary_crossentropy', optimizer=opt,
        metrics=['accuracy'])

    # Generate a print
    print('
    ------------------------------------------------------------------------
    ')
    print(f'Training for fold {fold_no} ...')

    # Fit data to model
    history = model.fit(X_train_class[train], y_train_class_count[
        train], epochs=100, verbose=0)
    #train_acc_per_fold.append(history.history['accuracy'])
    #train_loss_per_fold.append(history.history['loss'])
    #summarize_diagnostics(fold_no, history)
    model.save('porosity_bin_model_fold_no_'+ str(fold_no) +'.h5')

    # Generate generalization metrics
    train_scores = model.evaluate(X_train_class[train],
        y_train_class_count[train], verbose=0)
    print(f'Train Score for fold {fold_no}: {model.metrics_names
        [0]} of {train_scores[0]}; {model.metrics_names[1]} of {
        train_scores[1]*100}%')
    train_acc = train_scores[1]

    val_scores = model.evaluate(X_train_class[val],
        y_train_class_count[val], verbose=0)
    print(f'Val Score for fold {fold_no}: {model.metrics_names[0]}
        of {val_scores[0]}; {model.metrics_names[1]} of {val_scores
        [1]*100}%')
    val_acc = val_scores[1]

    test_scores = model.evaluate(X_test_class, y_test_class_count,
        steps=len(X_test_class), verbose = 0)
    print(f'Test data Score for fold {fold_no}: {model.
        metrics_names[0]} of {test_scores[0]}; {model.metrics_names
        [1]} of {test_scores[1]*100}%')
    test_acc = test_scores[1]

# Increase fold number
fold_no = fold_no + 1
```

```
    train_acc_per_fold.append(train_scores[1] * 100)
    train_loss_per_fold.append(train_scores[0])
    val_acc_per_fold.append(val_scores[1] * 100)
    val_loss_per_fold.append(val_scores[0])
    test_acc_per_fold.append(test_scores[1] * 100)
    test_loss_per_fold.append(test_scores[0])



X_test = NUMPY_INPUT_DATA[data_order[64:72],:,:,:]
y_test = y_class_count[data_order[64:72]]


LX = NUMPY_INPUT_DATA[data_order,:,:,:]
LY = y_class_count[data_order]

comb_count_acc = []
for i in range(1,9):
    model_name = 'porosity_bin_model_fold_no_' + str(i) + '.h5'
    model = load_model(model_name)
    pred = model.predict(LX)
    # LY = np.array([1 if a == 0 else 0 for a in Y])
    pred_class = [0 if a < 0.5 else 1 for a in pred]
    correct_X_ind = list(np.where(pred_class==LY)[0])
    print('
        ----------------------------------------------------------------------------------------------------
        ')
    print(f'Correct predictions in fold {i} ... ')
    print(len(correct_X_ind))
    pred_test = model.predict(X_test)
    pred_class_test = [0 if a < 0.5 else 1 for a in pred_test]
    correct_X_ind = list(np.where(pred_class_test==y_test)[0])
    print(f'Correct Test data predictions for fold {i} ... ')
    print(len(correct_X_ind))
    comb_count_acc.append(len(correct_X_ind)/len(X_test))


plt.plot(list(range(1,9)),comb_count_acc, label = 'Accuracy Per Fold')
plt.plot(list(range(1,9)),[np.mean(comb_count_acc) for i in range(8)],
   label = 'Average Accuracy')
plt.legend()
plt.title('Combined Model Accuracy in Predicting Pore Count',fontsize
   =12)
plt.ylabel('Accuracy',fontsize=14,labelpad=10)
plt.xlabel('Fold',fontsize=14)
plt.show()
```

```
NUMPY_INPUT_DATA_printing = TENSOR_INPUT_DATA_PRINTING.numpy()

X_train_class_printing = NUMPY_INPUT_DATA_printing[data_order
    [0:64],:,:,:]
y_train_class_count_printing = y_class_count[data_order[0:64]]
X_test_class_printing = NUMPY_INPUT_DATA_printing[data_order
    [64:72],:,:,:]
y_test_class_count_printing = y_class_count[data_order[64:72]]

# Define the K-fold Cross Validator
kfold = KFold(n_splits=8, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 1
train_acc_per_fold = []
train_loss_per_fold = []
val_acc_per_fold = []
val_loss_per_fold = []
test_acc_per_fold = []
test_loss_per_fold = []

for train, val in kfold.split(X_train_class_printing,
    y_train_class_count_printing):

    train_acc = 0
    val_acc = 0
    test_acc = 0
    # Define the model architecture
    while(train_acc < 0.5 or val_acc < 0.5 or test_acc < 0.5):
        model = Sequential()
        model.add(Conv2D(6, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same', input_shape
            =(4, 129, 15)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(1, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        # model.add(Conv2D(128, (3, 3), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        # model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(6, activation='relu', kernel_initializer='
            he_uniform'))
        model.add(Dense(1, activation='sigmoid'))

        # Compile the model
        opt = SGD(lr=0.001, momentum=0.9)
        model.compile(loss='binary_crossentropy', optimizer=opt,
```

```
        metrics=['accuracy'])

    # Generate a print
    print('
    ------------------------------------------------------------------------
    ')
    print(f'Training for fold {fold_no} ...')

    # Fit data to model
    history = model.fit(X_train_class_printing[train],
        y_train_class_count_printing[train], epochs=100, verbose=0)
    #train_acc_per_fold.append(history.history['accuracy'])
    #train_loss_per_fold.append(history.history['loss'])
    #summarize_diagnostics(fold_no, history)
    model.save('porosity_bin_model_printing_fold_no_'+ str(fold_no)
        +'.h5')

    # Generate generalization metrics
    train_scores = model.evaluate(X_train_class_printing[train],
        y_train_class_count_printing[train], verbose=0)
    print(f'Train Score for fold {fold_no}: {model.metrics_names
        [0]} of {train_scores[0]}; {model.metrics_names[1]} of {
        train_scores[1]*100}%')
    train_acc = train_scores[1]

    val_scores = model.evaluate(X_train_class_printing[val],
        y_train_class_count_printing[val], verbose=0)
    print(f'Val Score for fold {fold_no}: {model.metrics_names[0]}
        of {val_scores[0]}; {model.metrics_names[1]} of {val_scores
        [1]*100}%')
    val_acc = val_scores[1]

    test_scores = model.evaluate(X_test_class_printing,
        y_test_class_count_printing, steps=len(X_test_class_printing
        ), verbose = 0)
    print(f'Test data Score for fold {fold_no}: {model.
        metrics_names[0]} of {test_scores[0]}; {model.metrics_names
        [1]} of {test_scores[1]*100}%')
    test_acc = test_scores[1]

# Increase fold number
fold_no = fold_no + 1
train_acc_per_fold.append(train_scores[1] * 100)
train_loss_per_fold.append(train_scores[0])
val_acc_per_fold.append(val_scores[1] * 100)
val_loss_per_fold.append(val_scores[0])
test_acc_per_fold.append(test_scores[1] * 100)
test_loss_per_fold.append(test_scores[0])
```

```
LX = NUMPY_INPUT_DATA_printing[data_order,:,:,:]
LY = y_class_count[data_order]

printing_count_acc = []
for i in range(1,9):
    model_name = 'porosity_bin_model_printing_fold_no_' + str(i) + '.h5
        '
    model = load_model(model_name)
    pred = model.predict(LX)
    # LY = np.array([1 if a == 0 else 0 for a in Y])
    pred_class = [0 if a < 0.5 else 1 for a in pred]
    correct_X_ind = list(np.where(pred_class==LY)[0])
    print('
        ------------------------------------------------------------------------------------------------------------------------
        ')
    print(f'Correct predictions in fold {i} ... ')
    print(len(correct_X_ind))
    pred_test = model.predict(X_test_class_printing)
    pred_class_test = [0 if a < 0.5 else 1 for a in pred_test]
    correct_X_ind = list(np.where(pred_class_test==
        y_test_class_count_printing)[0])
    print(f'Correct Test data predictions for fold {i} ... ')
    print(len(correct_X_ind))
    printing_count_acc.append(len(correct_X_ind)/len(
        X_test_class_printing))


plt.plot(list(range(1,9)),printing_count_acc, label = 'Accuracy Per
    Fold')
plt.plot(list(range(1,9)),[np.mean(printing_count_acc) for i in range
    (8)], label = 'Average Accuracy')
plt.legend()
plt.title('Printing Model Accuracy in Predicting Pore Count',fontsize
    =12)
plt.ylabel('Accuracy',fontsize=14,labelpad=10)
plt.xlabel('Fold',fontsize=14)
plt.show()


NUMPY_INPUT_DATA_milling = TENSOR_INPUT_DATA_MILLING.numpy()

X_train_class_milling = NUMPY_INPUT_DATA_milling[data_order
    [0:64],:,:,:]
y_train_class_count_milling = y_class_count[data_order[0:64]]
X_test_class_milling = NUMPY_INPUT_DATA_milling[data_order
    [64:72],:,:,:]
```

```
y_test_class_count_milling = y_class_count[data_order[64:72]]

# Define the K-fold Cross Validator
kfold = KFold(n_splits=8, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 1
train_acc_per_fold = []
train_loss_per_fold = []
val_acc_per_fold = []
val_loss_per_fold = []
test_acc_per_fold = []
test_loss_per_fold = []

for train, val in kfold.split(X_train_class_milling,
    y_train_class_count_milling):

    train_acc = 0
    val_acc = 0
    test_acc = 0
    # Define the model architecture
    while(train_acc < 0.5 or val_acc < 0.5 or test_acc < 0.5):
        model = Sequential()
        model.add(Conv2D(6, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same', input_shape
            =(2, 129, 15)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(1, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        #model.add(MaxPooling2D((2, 2)))
        # model.add(Conv2D(128, (3, 3), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        # model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(6, activation='relu', kernel_initializer='
            he_uniform'))
        model.add(Dense(1, activation='sigmoid'))

        # Compile the model
        opt = SGD(lr=0.001, momentum=0.9)
        model.compile(loss='binary_crossentropy', optimizer=opt,
            metrics=['accuracy'])

        # Generate a print
        print('
            ------------------------------------------------------------------------
            ')
        print(f'Training for fold {fold_no} ...')
```

```
        # Fit data to model
        history = model.fit(X_train_class_milling[train],
            y_train_class_count_milling[train], epochs=100, verbose=0)
        #train_acc_per_fold.append(history.history['accuracy'])
        #train_loss_per_fold.append(history.history['loss'])
        #summarize_diagnostics(fold_no, history)
        model.save('porosity_bin_model_milling_fold_no_'+ str(fold_no)
            +'.h5')

        # Generate generalization metrics
        train_scores = model.evaluate(X_train_class_milling[train],
            y_train_class_count_milling[train], verbose=0)
        print(f'Train Score for fold {fold_no}: {model.metrics_names
            [0]} of {train_scores[0]}; {model.metrics_names[1]} of {
            train_scores[1]*100}%')
        train_acc = train_scores[1]

        val_scores = model.evaluate(X_train_class_milling[val],
            y_train_class_count_milling[val], verbose=0)
        print(f'Val Score for fold {fold_no}: {model.metrics_names[0]}
            of {val_scores[0]}; {model.metrics_names[1]} of {val_scores
            [1]*100}%')
        val_acc = val_scores[1]

        test_scores = model.evaluate(X_test_class_milling,
            y_test_class_count_milling, steps=len(X_test_class_milling),
             verbose = 0)
        print(f'Test data Score for fold {fold_no}: {model.
            metrics_names[0]} of {test_scores[0]}; {model.metrics_names
            [1]} of {test_scores[1]*100}%')
        test_acc = test_scores[1]

    # Increase fold number
    fold_no = fold_no + 1
    train_acc_per_fold.append(train_scores[1] * 100)
    train_loss_per_fold.append(train_scores[0])
    val_acc_per_fold.append(val_scores[1] * 100)
    val_loss_per_fold.append(val_scores[0])
    test_acc_per_fold.append(test_scores[1] * 100)
    test_loss_per_fold.append(test_scores[0])


LX = NUMPY_INPUT_DATA_milling[data_order,:,:,:]
LY = y_class_count[data_order]

milling_count_acc = []
for i in range(1,9):
```

```
    model_name = 'porosity_bin_model_milling_fold_no_' + str(i) + '.h5'
    model = load_model(model_name)
    pred = model.predict(LX)
    # LY = np.array([1 if a == 0 else 0 for a in Y])
    pred_class = [0 if a < 0.5 else 1 for a in pred]
    correct_X_ind = list(np.where(pred_class==LY)[0])
    print('
    ----------------------------------------------------------------------
    ')
    print(f'Correct predictions in fold {i} ...')
    print(len(correct_X_ind))
    pred_test = model.predict(X_test_class_milling)
    pred_class_test = [0 if a < 0.5 else 1 for a in pred_test]
    correct_X_ind = list(np.where(pred_class_test==
        y_test_class_count_milling)[0])
    print(f'Correct Test data predictions for fold {i} ...')
    print(len(correct_X_ind))
    milling_count_acc.append(len(correct_X_ind)/len(
        X_test_class_milling))


plt.plot(list(range(1,9)), milling_count_acc, label = 'Accuracy Per
    Fold')
plt.plot(list(range(1,9)),[np.mean( milling_count_acc) for i in range
    (8)], label = 'Average Accuracy')
plt.legend()
plt.title('Milling Model Accuracy in Predicting Pore Count',fontsize
    =12)
plt.ylabel('Accuracy',fontsize=14,labelpad=10)
plt.xlabel('Fold',fontsize=14)
plt.show()


y_class_area = np.zeros(72)
for i in range(72):
  if y_target_area[i]<=np.mean(y_target_area):
    y_class_area[i] = 0
  else:
    y_class_area[i] = 1


# CNN Model Architecture - Printing and Milling combined cycles

NUMPY_INPUT_DATA = TENSOR_INPUT_DATA_COMBINED.numpy()


X_train_class = NUMPY_INPUT_DATA[data_order[0:64],:,:,:]
y_train_class_area = y_class_area[data_order[0:64]]
```

```
X_test_class = NUMPY_INPUT_DATA[data_order[64:72],:,:,:]
y_test_class_area = y_class_area[data_order[64:72]]


# Define the K-fold Cross Validator
kfold = KFold(n_splits=8, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 1
train_acc_per_fold = []
train_loss_per_fold = []
val_acc_per_fold = []
val_loss_per_fold = []
test_acc_per_fold = []
test_loss_per_fold = []

for train, val in kfold.split(X_train_class, y_train_class_area):

    train_acc = 0
    val_acc = 0
    test_acc = 0
    # Define the model architecture
    while(train_acc < 0.5 or val_acc < 0.5 or test_acc < 0.5):
        model = Sequential()
        model.add(Conv2D(6, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same', input_shape
            =(6, 129, 15)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(1, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        # model.add(Conv2D(128, (3, 3), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        # model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(6, activation='relu', kernel_initializer='
            he_uniform'))
        model.add(Dense(1, activation='sigmoid'))

        # Compile the model
        opt = SGD(lr=0.001, momentum=0.9)
        model.compile(loss='binary_crossentropy', optimizer=opt,
            metrics=['accuracy'])

        # Generate a print
        print('
            ------------------------------------------------------------------------
            ')
```

```
        print(f'Training for fold {fold_no} ...')

        # Fit data to model
        history = model.fit(X_train_class[train], y_train_class_area[
            train], epochs=100, verbose=0)
        #train_acc_per_fold.append(history.history['accuracy'])
        #train_loss_per_fold.append(history.history['loss'])
        #summarize_diagnostics(fold_no, history)
        model.save('porosity_bin_model_fold_no_'+ str(fold_no) +'.h5')

        # Generate generalization metrics
        train_scores = model.evaluate(X_train_class[train],
            y_train_class_area[train], verbose=0)
        print(f'Train Score for fold {fold_no}: {model.metrics_names
            [0]} of {train_scores[0]}; {model.metrics_names[1]} of {
            train_scores[1]*100}%')
        train_acc = train_scores[1]

        val_scores = model.evaluate(X_train_class[val],
            y_train_class_area[val], verbose=0)
        print(f'Val Score for fold {fold_no}: {model.metrics_names[0]}
            of {val_scores[0]}; {model.metrics_names[1]} of {val_scores
            [1]*100}%')
        val_acc = val_scores[1]

        test_scores = model.evaluate(X_test_class, y_test_class_area,
            steps=len(X_test_class), verbose = 0)
        print(f'Test data Score for fold {fold_no}: {model.
            metrics_names[0]} of {test_scores[0]}; {model.metrics_names
            [1]} of {test_scores[1]*100}%')
        test_acc = test_scores[1]

    # Increase fold number
    fold_no = fold_no + 1
    train_acc_per_fold.append(train_scores[1] * 100)
    train_loss_per_fold.append(train_scores[0])
    val_acc_per_fold.append(val_scores[1] * 100)
    val_loss_per_fold.append(val_scores[0])
    test_acc_per_fold.append(test_scores[1] * 100)
    test_loss_per_fold.append(test_scores[0])

LX = NUMPY_INPUT_DATA[data_order,:,:,:]
LY = y_class_area[data_order]

comb_area_acc = []
for i in range(1,9):
    model_name = 'porosity_bin_model_fold_no_' + str(i) + '.h5'
    model = load_model(model_name)
```

```
    pred = model.predict(LX)
    # LY = np.array([1 if a == 0 else 0 for a in Y])
    pred_class = [0 if a < 0.5 else 1 for a in pred]
    correct_X_ind = list(np.where(pred_class==LY)[0])
    print('
    ----------------------------------------------------------------------
    ')
    print(f'Correct predictions in fold {i} ...')
    print(len(correct_X_ind))
    pred_test = model.predict(X_test)
    pred_class_test = [0 if a < 0.5 else 1 for a in pred_test]
    correct_X_ind = list(np.where(pred_class_test==y_test_class_area)
    [0])
    print(f'Correct Test data predictions for fold {i} ...')
    print(len(correct_X_ind))
    comb_area_acc.append(len(correct_X_ind)/len(X_test_class_milling))


plt.plot(list(range(1,9)), comb_area_acc, label = 'Accuracy Per Fold')
plt.plot(list(range(1,9)),[np.mean( comb_area_acc) for i in range(8)],
   label = 'Average Accuracy')
plt.legend()
plt.title('Combined Model Accuracy in Predicting %Area Porosity',
   fontsize=12)
plt.ylabel('Accuracy',fontsize=14,labelpad=10)
plt.xlabel('Fold',fontsize=14)
plt.show()


NUMPY_INPUT_DATA_printing = TENSOR_INPUT_DATA_PRINTING.numpy()

X_train_printing = NUMPY_INPUT_DATA_printing[data_order[0:64],:,:,:]
y_train_area_printing = y_class_area[data_order[0:64]]
X_test_printing = NUMPY_INPUT_DATA_printing[data_order[64:72],:,:,:]
y_test_area_printing = y_class_area[data_order[64:72]]

# Define the K-fold Cross Validator
kfold = KFold(n_splits=8, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 1
train_acc_per_fold = []
train_loss_per_fold = []
val_acc_per_fold = []
val_loss_per_fold = []
test_acc_per_fold = []
test_loss_per_fold = []
```

```
for train, val in kfold.split(X_train_printing, y_train_area_printing):

    train_acc = 0
    val_acc = 0
    test_acc = 0
    # Define the model architecture
    while(train_acc < 0.5 or val_acc < 0.5 or test_acc < 0.5):
        model = Sequential()
        model.add(Conv2D(6, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same', input_shape
            =(4, 129, 15)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(1, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        # model.add(Conv2D(128, (3, 3), activation='relu',
            kernel_initializer='he_uniform', padding='same'))
        # model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(6, activation='relu', kernel_initializer='
            he_uniform'))
        model.add(Dense(1, activation='sigmoid'))

        # Compile the model
        opt = SGD(lr=0.001, momentum=0.9)
        model.compile(loss='binary_crossentropy', optimizer=opt,
            metrics=['accuracy'])

        # Generate a print
        print('
            ------------------------------------------------------------------------
            ')
        print(f'Training for fold {fold_no} ...')

        # Fit data to model
        history = model.fit(X_train_printing[train],
            y_train_area_printing[train], epochs=100, verbose=0)
        #train_acc_per_fold.append(history.history['accuracy'])
        #train_loss_per_fold.append(history.history['loss'])
        #summarize_diagnostics(fold_no, history)
        model.save('porosity_bin_model_printing_fold_no_'+ str(fold_no)
            +'.h5')

        # Generate generalization metrics
        train_scores = model.evaluate(X_train_printing[train],
            y_train_area_printing[train], verbose=0)
        print(f'Train Score for fold {fold_no}: {model.metrics_names
            [0]} of {train_scores[0]}; {model.metrics_names[1]} of {
```

```
            train_scores[1]*100}%')
        train_acc = train_scores[1]

        val_scores = model.evaluate(X_train_printing[val],
            y_train_area_printing[val], verbose=0)
        print(f'Val Score for fold {fold_no}: {model.metrics_names[0]}
            of {val_scores[0]}; {model.metrics_names[1]} of {val_scores
            [1]*100}%')
        val_acc = val_scores[1]

        test_scores = model.evaluate(X_test_printing,
            y_test_area_printing, steps=len(X_test_printing), verbose =
            0)
        print(f'Test data Score for fold {fold_no}: {model.
            metrics_names[0]} of {test_scores[0]}; {model.metrics_names
            [1]} of {test_scores[1]*100}%')
        test_acc = test_scores[1]

    # Increase fold number
    fold_no = fold_no + 1
    train_acc_per_fold.append(train_scores[1] * 100)
    train_loss_per_fold.append(train_scores[0])
    val_acc_per_fold.append(val_scores[1] * 100)
    val_loss_per_fold.append(val_scores[0])
    test_acc_per_fold.append(test_scores[1] * 100)
    test_loss_per_fold.append(test_scores[0])


LX = NUMPY_INPUT_DATA_printing[data_order,:,:,:]
LY = y_class_area[data_order]

printing_area_acc = []
for i in range(1,9):
    model_name = 'porosity_bin_model_printing_fold_no_' + str(i) + '.h5
        '
    model = load_model(model_name)
    pred = model.predict(LX)
    # LY = np.array([1 if a == 0 else 0 for a in Y])
    pred_class = [0 if a < 0.5 else 1 for a in pred]
    correct_X_ind = list(np.where(pred_class==LY)[0])
    print('
        ------------------------------------------------------------------------------------------------------------------------
        ')
    print(f'Correct predictions in fold {i} ...')
    print(len(correct_X_ind))
    pred_test = model.predict(X_test_printing)
    pred_class_test = [0 if a < 0.5 else 1 for a in pred_test]
    correct_X_ind = list(np.where(pred_class_test==y_test_area_printing
```

```
        )[0])
    print(f'Correct Test data predictions for fold {i} ...')
    print(len(correct_X_ind))
    printing_area_acc.append(len(correct_X_ind)/len(
        X_test_class_milling))


plt.plot(list(range(1,9)), printing_area_acc, label = 'Accuracy Per
    Fold')
plt.plot(list(range(1,9)),[np.mean( printing_area_acc) for i in range
    (8)], label = 'Average Accuracy')
plt.legend()
plt.title('Printing Model Accuracy in Predicting %Area Porosity',
    fontsize=12)
plt.ylabel('Accuracy',fontsize=14,labelpad=10)
plt.xlabel('Fold',fontsize=14)
plt.show()

NUMPY_INPUT_DATA_milling = TENSOR_INPUT_DATA_MILLING.numpy()

X_train_milling = NUMPY_INPUT_DATA_milling[data_order[0:64],:,:,:]
y_train_area_milling = y_class_area[data_order[0:64]]
X_test_milling = NUMPY_INPUT_DATA_milling[data_order[64:72],:,:,:]
y_test_area_milling = y_class_area[data_order[64:72]]

# Define the K-fold Cross Validator
kfold = KFold(n_splits=8, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 1
train_acc_per_fold = []
train_loss_per_fold = []
val_acc_per_fold = []
val_loss_per_fold = []
test_acc_per_fold = []
test_loss_per_fold = []

for train, val in kfold.split(X_train_milling, y_train_area_milling):

    train_acc = 0
    val_acc = 0
    test_acc = 0
    # Define the model architecture
    while(train_acc < 0.5 or val_acc < 0.5 or test_acc < 0.5):
        model = Sequential()
        model.add(Conv2D(6, (7, 7), activation='relu',
            kernel_initializer='he_uniform', padding='same', input_shape
            =(2, 129, 15)))
```

```
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(1, (7, 7), activation='relu',
    kernel_initializer='he_uniform', padding='same'))
#model.add(MaxPooling2D((2, 2)))
# model.add(Conv2D(128, (3, 3), activation='relu',
    kernel_initializer='he_uniform', padding='same'))
# model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(6, activation='relu', kernel_initializer='
    he_uniform'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
opt = SGD(lr=0.001, momentum=0.9)
model.compile(loss='binary_crossentropy', optimizer=opt,
    metrics=['accuracy'])

# Generate a print
print('
    ------------------------------------------------------------------------
    ')
print(f'Training for fold {fold_no} ...')

# Fit data to model
history = model.fit(X_train_milling[train],
    y_train_area_milling[train], epochs=100, verbose=0)
#train_acc_per_fold.append(history.history['accuracy'])
#train_loss_per_fold.append(history.history['loss'])
#summarize_diagnostics(fold_no, history)
model.save('porosity_bin_model_milling_fold_no_'+ str(fold_no)
    +'.h5')

# Generate generalization metrics
train_scores = model.evaluate(X_train_milling[train],
    y_train_area_milling[train], verbose=0)
print(f'Train Score for fold {fold_no}: {model.metrics_names
    [0]} of {train_scores[0]}; {model.metrics_names[1]} of {
    train_scores[1]*100}%')
train_acc = train_scores[1]

val_scores = model.evaluate(X_train_milling[val],
    y_train_area_milling[val], verbose=0)
print(f'Val Score for fold {fold_no}: {model.metrics_names[0]}
    of {val_scores[0]}; {model.metrics_names[1]} of {val_scores
    [1]*100}%')
val_acc = val_scores[1]

test_scores = model.evaluate(X_test_milling,
```

```
            y_test_area_milling, steps=len(X_test), verbose = 0)
        print(f'Test data Score for fold {fold_no}: {model.
            metrics_names[0]} of {test_scores[0]}; {model.metrics_names
            [1]} of {test_scores[1]*100}%')
        test_acc = test_scores[1]

    # Increase fold number
    fold_no = fold_no + 1
    train_acc_per_fold.append(train_scores[1] * 100)
    train_loss_per_fold.append(train_scores[0])
    val_acc_per_fold.append(val_scores[1] * 100)
    val_loss_per_fold.append(val_scores[0])
    test_acc_per_fold.append(test_scores[1] * 100)
    test_loss_per_fold.append(test_scores[0])


LX = NUMPY_INPUT_DATA_milling[data_order,:,:,:]
LY = y_class_area[data_order]

milling_area_acc = []
for i in range(1,9):
    model_name = 'porosity_bin_model_milling_fold_no_' + str(i) + '.h5'
    model = load_model(model_name)
    pred = model.predict(LX)
    # LY = np.array([1 if a == 0 else 0 for a in Y])
    pred_class = [0 if a < 0.5 else 1 for a in pred]
    correct_X_ind = list(np.where(pred_class==LY)[0])
    print('
        ----------------------------------------------------------------------
        ')
    print(f'Correct predictions in fold {i} ...')
    print(len(correct_X_ind))
    pred_test = model.predict(X_test_milling)
    pred_class_test = [0 if a < 0.5 else 1 for a in pred_test]
    correct_X_ind = list(np.where(pred_class_test==y_test_area_milling)
        [0])
    print(f'Correct Test data predictions for fold {i} ...')
    print(len(correct_X_ind))
    milling_area_acc.append(len(correct_X_ind)/len(X_test_class_milling
        ))


plt.plot(list(range(1,9)), milling_area_acc, label = 'Accuracy Per Fold
    ')
plt.plot(list(range(1,9)),[np.mean( milling_area_acc) for i in range(8)
    ], label = 'Average Accuracy')
plt.legend()
plt.title('Milling Model Accuracy in Predicting %Area Porosity',
```

```
    fontsize=12)
plt.ylabel('Accuracy',fontsize=14,labelpad=10)
plt.xlabel('Fold',fontsize=14)
plt.show()
```