**PROJECT REPORT**

**ON**

“Understanding of the parameters optimization techniques used in metal additive manufacturing process”

**BY**

Vansh Maheshwari

Under the supervision of

**DR. RAVI VIDHYARTHY**

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

OF

MEF266: STUDY PROJECT

****

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI**

**HYDERABAD CAMPUS**

**(MAY 2024)**

**ACKNOWLEDGMENT**

We would like to express our sincere gratitude to Dr. Ravi Vidhyarthy for providing us the opportunity to work on the project " **Understanding of the parameters optimization techniques used in metal additive manufacturing process**" and giving us his constant mentoring, without which we would not have been able to advance in our project. We would like to thank Sai Kiran sir for his help in the workshop.

We would like to thank Mechanical Department, BPHC for inspiring us to do this project.

We appreciate the college administration’s assistance in ensuring easy access to a top-notch educational opportunity and learning experience.

****

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI**

**HYDERABAD CAMPUS**

**CERTIFICATE**

This is to certify that the project report entitled “**Bead geometry prediction for robotic GMAW-based rapid manufacturing through a neural network and a second-order regression analysis”** submitted by Mr. Vansh Maheshwari (ID No. 2021A4PS2534H) partially fulfills the requirements of the course ME F266 Study Project Course, embodies the work done by him under my supervision and guidance.

Date: 8 May 2024 (Dr. Ravi Vidhyarthy)

BITS- Pilani, Hyderabad Campus

**ABSTRACT**

This project aims to predict the bead geometry for robotic GMAW (Gas metal arc welding) - based rapid manufacturing through neural network (ANN), second-order regression analysis and XGBoost models

The project aims to build various machine learning models and compare them based on their accuracy and other performance parameters. Also, to establish accurate relationships between the input variables and the outputs in the design matrix and actual test results, error analysis is also done.

**CONTENTS**

Title …………………………………………………………………………....1 Acknowledgement…………………………………………………………….2 Certificate……………………………………………………...……………....3 Abstract………………………………………………………………………...4 Introduction……………………..…………….…….…….…………………...6

Design Matrix…………………………..…………………………………......7

Regression……..………………………………………………………...…...9

ANN…....................................................................................................13XGBoost.................................................................................................17

Workshop Job...…..….….………............................................................20

Conclusion.............................................................................................24

References............................................................................................24

**INTRODUCTION**

Additive manufacturing or 3D printing is adding layers of material to create a 3D object. This is different from traditional manufacturing methods, such as subtractive machining. It enables the creation of geometrical structures that may otherwise be difficult or impossible to create using traditional manufacturing processes.

The various steps involved in additive manufacturing are:

1. Designing (Using CAD software)

2. Slicing of the design into thin layers using suitable slicing software

3. Using the Additive manufacturing machine to deposit material layer by layer to build up the final object. The materials used for manufacturing include plastics, metals, ceramics, biological materials

4. End steps include curing, polishing, or surface finishing

There are several additive manufacturing (AM) technologies, each with its own principles, materials, and applications. Here are some of the common types:

1. Fused Deposition Modeling (FDM): one of the most widely used 3D printing technologies. It involves extruding thermoplastic material layer by layer through a heated nozzle.

2. Stereolithography (SLA): SLA uses a laser to solidify liquid resin layer by layer, building up the object.

3. Selective Laser Sintering (SLS): SLS uses a laser to sinter powdered material (typically nylon or other polymers) layer by layer.

4. Digital Light Processing (DLP): Similar to SLA, DLP uses light to cure liquid resin but does so with a digital light projector that exposes an entire layer simultaneously.

5. Selective Laser Melting (SLM): SLM melts metal powder layer by layer using a high-powered laser.

6. Electron Beam Melting (EBM): Similar to SLM but uses an electron beam to melt metal powder.

7. Binder Jetting: Binder jetting involves depositing a liquid binding agent onto a powder bed layer by layer.

**Why Bead Geometry is important?**

Bead geometry is critical in Additive Manufacturing (AM) because it directly affects:

* Strength: Proper bead shape ensures strong bonds between layers, producing a robust final part.
* Accuracy: Precise bead size helps achieve the intended dimensions of the printed object.
* Surface finish: Consistent bead geometry minimizes gaps and unevenness, resulting in a smoother surface.
* Material properties: Bead size and printing parameters influence the printed material's density, porosity, and anisotropy.
* Efficiency: Optimized bead geometry reduces material waste and printing time.

**Models to predict bead geometry:**

We are using the following models to predict the bead geometry:

1. ANN: An artificial neural network is a type of machine learning model inspired by the structure and functions of the human brain. ANNs consist of interconnected nodes called artificial neurons. They are loosely based on biological neurons and are organized in layers. Information flows between these layers. ANNs learn by adjusting the strengths of connections between neurons. This is similar to how synapses work in the brain. By adapting these connections, the network can learn to recognize patterns in the data. ANNs are powerful tools that can be used for a variety of tasks such as image recognition, speech recognition, and natural language processing. Basically, ANNs are a way for computers to learn and process information just like humans.
2. XGBoost: XGBoost (Extreme Gradient Boosting) provides a powerful and versatile tool for building high-performing machine learning models.

XGBoost's Advantages:

* Speed: XGBoost excels at handling large datasets efficiently.
* Accuracy: By combining learners, XGBoost achieves impressive prediction accuracy.
* Flexibility: It can work with various data types and even handle missing values effectively.

1. Second-order regression: The second-order regression model is a way to capture curved relationships between a predictor variable and a dependent variable. It is a step up from linear regression when you need to capture more complex, curved relationships in your data. Unlike a straight line in linear regression, a second-order model can bend upwards or downwards to fit a curved pattern in the data

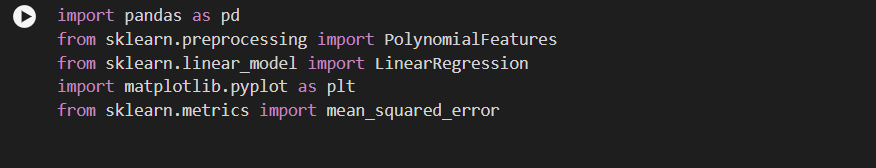
**DESIGN MATRIX**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. no** | **F**  **(m/min)** | **S (cm/min)** | **V**  **(V)** | **D**  **(mm)** | **W**  **(mm)** | **H**  **(mm)** |
| 1 | 3.6 | 22.50 | 17.50 | 9.00 | 8.95 | 2.88 |
| 2 | 5.20 | 22.50 | 17.50 | 9.00 | 10.72 | 3.35 |
| 3 | 3.60 | 37.50 | 17.50 | 9.00 | 7.19 | 2.45 |
| 4 | 5.20 | 37.50 | 17.50 | 9.00 | 8.29 | 2.75 |
| 5 | 3.60 | 22.50 | 20.50 | 9.00 | 10.25 | 2.66 |
| 6 | 5.20 | 22.50 | 20.50 | 9.00 | 11.5 | 3.26 |
| 7 | 3.60 | 37.50 | 20.50 | 9.00 | 8.36 | 2.17 |
| 8 | 5.20 | 37.50 | 20.50 | 9.00 | 9.35 | 2.58 |
| 9 | 3.60 | 22.50 | 17.50 | 15.00 | 8.36 | 3 |
| 10 | 5.20 | 22.50 | 17.50 | 15.00 | 9.52 | 3.56 |
| 11 | 3.60 | 37.50 | 17.50 | 15.00 | 6.83 | 2.45 |
| 12 | 5.20 | 37.50 | 17.50 | 15.00 | 7.98 | 2.9 |
| 13 | 3.60 | 22.50 | 20.50 | 15.00 | 9.92 | 2.79 |
| 14 | 5.20 | 22.50 | 20.50 | 15.00 | 11.12 | 3.35 |
| 15 | 3.60 | 37.50 | 20.50 | 15.00 | 7.91 | 2.26 |
| 16 | 5.20 | 37.50 | 20.50 | 15.00 | 9.25 | 2.7 |
| 17 | 2.80 | 30.00 | 19.00 | 12.00 | 7.39 | 2.32 |
| 18 | 6.00 | 30.00 | 19.00 | 12.00 | 9.9 | 3.28 |
| 19 | 4.40 | 15.00 | 19.00 | 12.00 | 11.76 | 3.8 |
| 20 | 4.40 | 45.00 | 19.00 | 12.00 | 7.54 | 2.34 |
| 21 | 4.40 | 30.00 | 16.00 | 12.00 | 8.08 | 2.94 |
| 22 | 4.40 | 30.00 | 22.00 | 12.00 | 9.9 | 2.45 |
| 23 | 4.40 | 30.00 | 19.00 | 6.00 | 9.51 | 2.77 |
| 24 | 4.40 | 30.00 | 19.00 | 18.00 | 8.58 | 2.83 |
| 25 | 4.40 | 30.00 | 19.00 | 12.00 | 8.88 | 2.75 |
| 26 | 4.40 | 30.00 | 19.00 | 12.00 | 9.09 | 2.83 |
| 27 | 4.40 | 30.00 | 19.00 | 12.00 | 8.92 | 2.79 |
| 28 | 4.40 | 30.00 | 19.00 | 12.00 | 8.91 | 2.75 |
| 29 | 4.40 | 30.00 | 19.00 | 12.00 | 8.92 | 2.83 |
| 30 | 4.40 | 30.00 | 19.00 | 12.00 | 9.02 | 2.81 |
| 31 | 4.40 | 30.00 | 19.00 | 12.00 | 8.8 | 2.8 |

**Model 1: Second-order regression**

Steps involved:

1.Importing necessary Python libraries: Pandas for data manipulation and reading excel files, Polynomial Features for generating polynomial features, Linear Regression for building linear regression models, matplotlib. pyplot for plotting results, mean\_squared\_error from sklearn. metrics to evaluate model performance and error analysis and NumPy for numerical operations and creating a custom error metric



2.Loading data from excel:

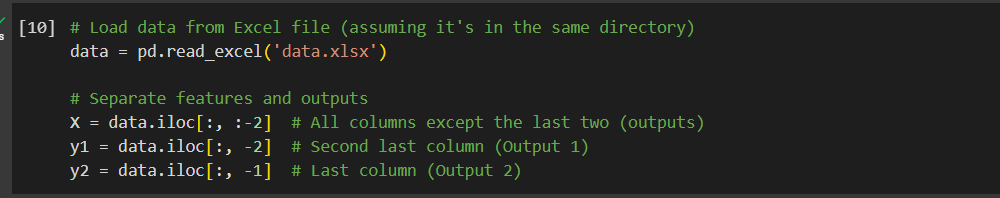
data = pd.read\_excel('data.xlsx') reads data from an Excel file (ensure it's in the same directory).

3.Separate features and outputs:

X = data.iloc[:, :-2] extracts all columns except the last two as features.

y1 = data.iloc[:, -2] extracts the second last column as the first output (W in mm).

y2 = data.iloc[:, -1] extracts the last column as the second output (H in mm).

 4.Create Polynomial Features:

degree = 2 sets the degree for polynomial transformation.

poly = Polynomial Features creates a PolynomialFeatures object.

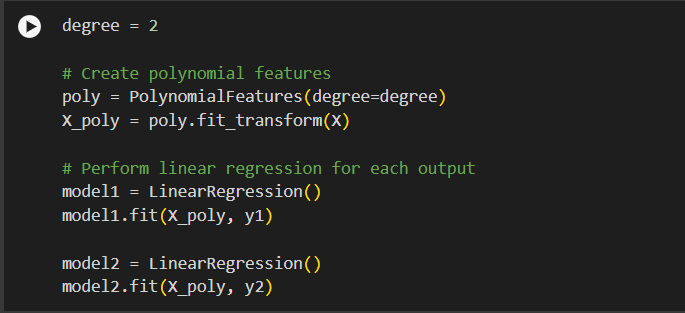
X\_poly = poly.fit\_transform(X) transforms the features into polynomial features (including squares and interactions).

5.Build Linear Regression Models:

model1 = LinearRegression() creates a linear regression model for Output 1.

model1.fit(X\_poly, y1) trains the model1 on transformed features and Output 1.

Repeat for Output 2: model2 = LinearRegression() and model2.fit(X\_poly, y2).



6.Make Predictions and Evaluate Performance:

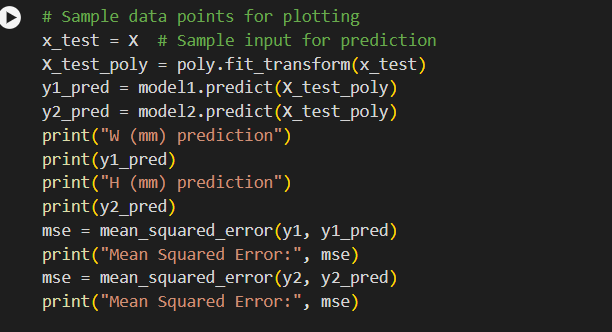
x\_test for sample predictions (using original features).

X\_test\_poly = poly.fit\_transform(x\_test) transforms sample features.

y1\_pred = model1.predict(X\_test\_poly) predicts Output 1 using model1.

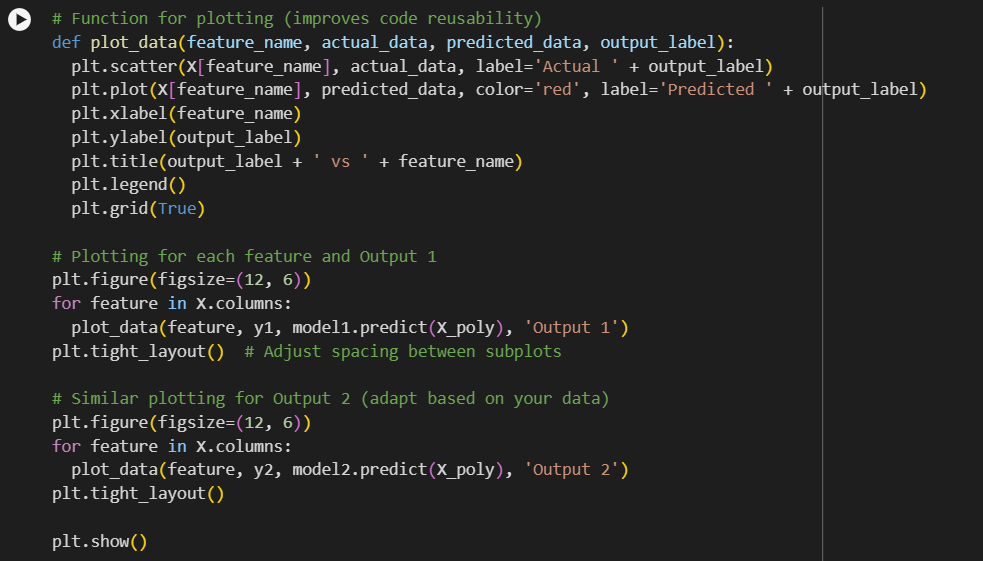
y2\_pred = model2.predict(X\_test\_poly) predicts Output 2 using model2.

Calculate and print Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE) for both outputs.

7.Plot Results for Validation:

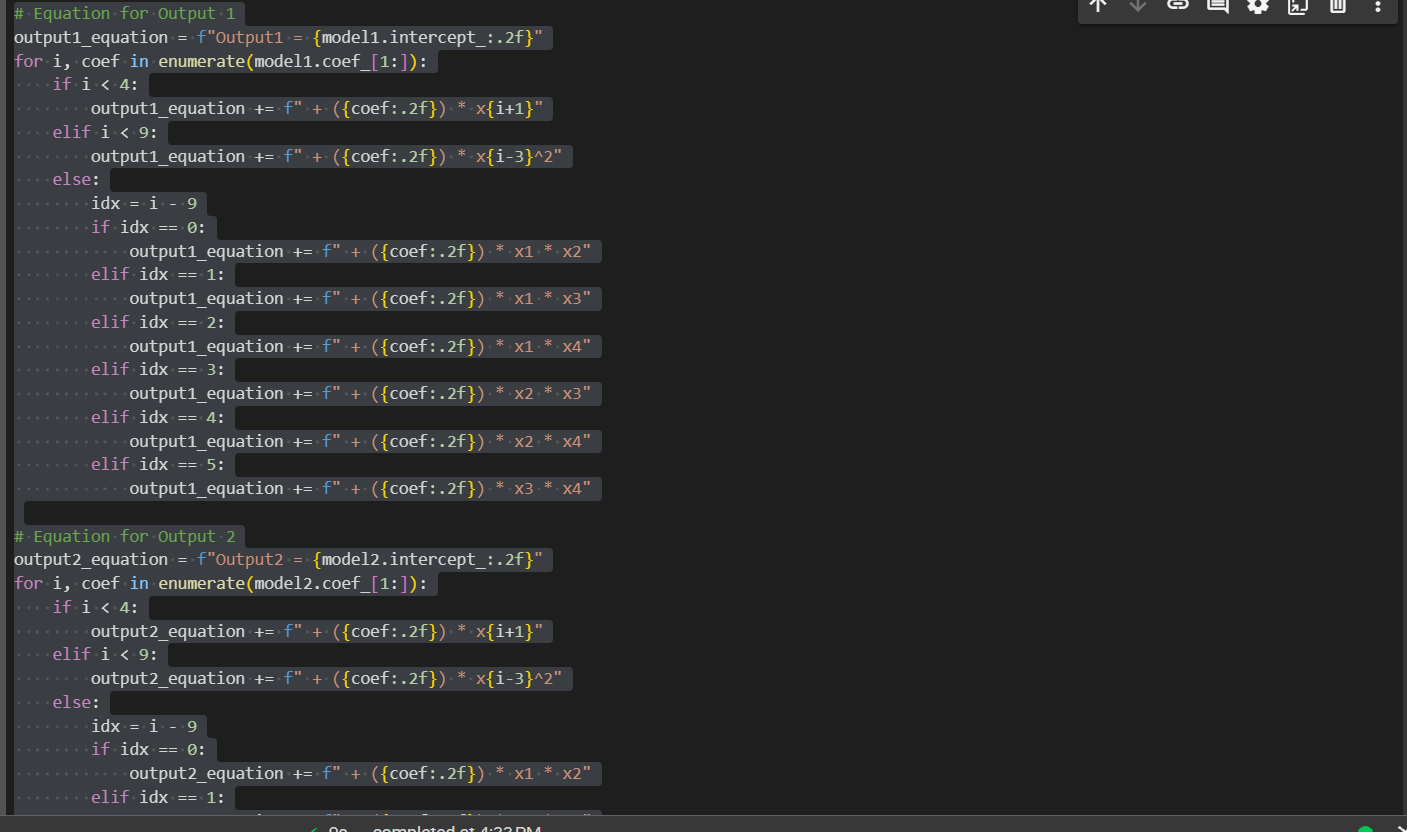
plot\_data function for reusable plotting.

Plot actual vs. predicted values for Output 1 and Output 2 against each feature.



8.Display Model Equations:

Construct equations for Output 1 and Output 2 using model coefficients and intercepts.



Outputs:

1.The predicted values of W and H are as follows:

| W (mm) | H (mm) |
| --- | --- |
| 9.061168 | 2.880611 |
| 10.48866 | 3.370609 |
| 7.186147 | 2.418107 |
| 8.413638 | 2.760606 |
| 10.271126 | 2.675604 |
| 11.598617 | 3.223102 |
| 8.231105 | 2.1856 |
| 9.358596 | 2.585598 |
| 8.31859 | 2.995597 |
| 9.681081 | 3.543096 |
| 6.763568 | 2.485594 |
| 7.92606 | 2.885592 |
| 9.828548 | 2.77809 |
| 11.091039 | 3.383089 |
| 8.108526 | 2.240586 |
| 9.171017 | 2.698085 |
| 7.390297 | 2.320051 |
| 9.900332 | 3.280057 |
| 11.760271 | 3.800047 |
| 7.540357 | 2.340062 |
| 8.08023 | 2.94004 |
| 9.900399 | 2.450069 |
| 9.510157 | 2.770027 |
| 8.580471 | 2.830081 |
| 8.975869 | 2.779632 |
| 8.963362 | 2.783905 |
| 8.950043 | 2.788546 |
| 8.935911 | 2.793553 |
| 8.920967 | 2.798926 |
| 8.905209 | 2.804666 |
| 8.888639 | 2.810772 |

2.Error Analysis:

MAPE of the predicted values:

Mean Absolute Percentage Error for W (mm): 0.6917

Mean Absolute Percentage Error for H (mm): 0.5136

3.Equations used:

For W(mm): Output1 = 5.49 + (0.19) \* x1 + (2.77) \* x2 + (-0.30) \* x3 + (0.28) \* x4 + (-0.78) \* x1^2 + (-0.00) \* x2^2 + (0.00) \* x3^2 + (-0.00) \* x4^2 + (-0.01) \* x5^2 + (0.00) \* x1 \* x2 + (-0.16) \* x1 \* x3 + (-0.01) \* x1 \* x4 + (-0.02) \* x2 \* x3 + (-0.01) \* x2 \* x4 + (0.00) \* x3 \* x4

For H(mm): Output2 = 3.70 + (0.06) \* x1 + (0.02) \* x2 + (-0.11) \* x3 + (0.14) \* x4 + (-0.08) \* x1^2 + (0.00) \* x2^2 + (0.00) \* x3^2 + (-0.00) \* x4^2 + (-0.00) \* x5^2 + (-0.00) \* x1 \* x2 + (0.02) \* x1 \* x3 + (-0.01) \* x1 \* x4 + (0.01) \* x2 \* x3 + (0.01) \* x2 \* x4 + (0.00) \* x3 \* x4

Here X1=F, X2=S, X3=V, X4=D

**Model 2: ANN**

1. Importing Libraries:

numpy: Numerical operations.

pandas: Data manipulation (reading CSV files, creating DataFrames).

matplotlib.pyplot: Data visualization (plotting).

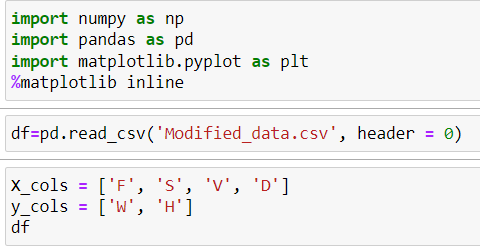
2. Load Data:

df = pd.read\_csv('Modified\_data.csv', header=0) reads data from a CSV file assuming it has a header row.

3. Define Feature and Output Columns:

X\_cols = ['F', 'S', 'V', 'D'] defines the feature column names.

y\_cols = ['W', 'H'] defines the output column names.



4. Separate Features and Outputs:

X = df[X\_cols] creates a DataFrame containing the features.

y = df[y\_cols] creates a DataFrame containing the outputs.

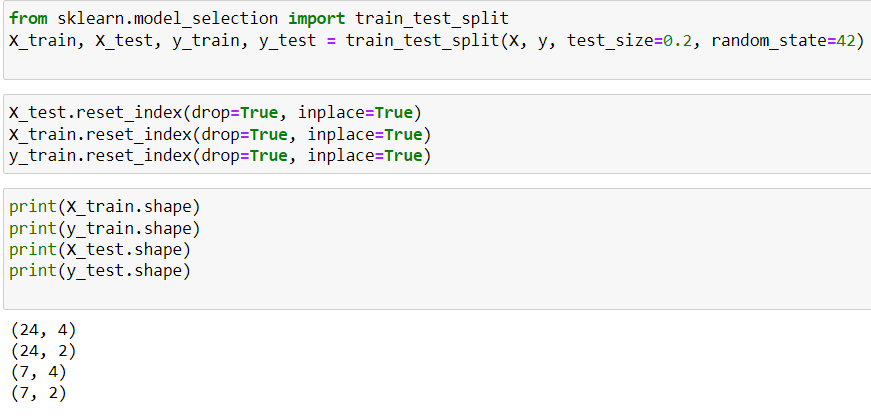
5. Split Data (Train-Test):

from sklearn.model\_selection import train\_test\_split imports the train\_test\_split function for data splitting. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) splits the data into training (80%) and testing (20%) sets with a random seed of 42 for reproducibility.

reset\_index(drop=True, inplace=True) on training and testing sets removes unnecessary index columns.

6. Print Data Shapes:

Prints the shapes of training and testing sets to verify their dimensions.



7. Import Preprocessing and Model Building Libraries:

from sklearn.preprocessing import StandardScaler imports the StandardScaler for feature scaling.

import tensorflow as tf imports TensorFlow for building the neural network.

from tensorflow.keras.models import Sequential imports the Sequential model architecture.

from tensorflow.keras.layers import Dense imports the Dense layer for building the network.

8. Feature Scaling:

scaler = StandardScaler() creates a StandardScaler object.

X\_train = scaler.fit\_transform(X\_train) scales the training features (ensures features are on a similar scale).

X\_test = scaler.transform(X\_test) scales the testing features using the scaler fitted on the training data.

9. Build Neural Network Model:

model = tf.keras.models.Sequential() creates a sequential neural network model.

model.add(Dense(units=16, input\_dim=4, kernel\_initializer='normal', activation='relu')):

Adds a Dense layer with 16 neurons as the first hidden layer.

input\_dim=4 specifies the input dimension (number of features).

kernel\_initializer='normal' initializes weights with a normal distribution.

activation='relu' uses the ReLU activation function.

Adds two more hidden layers with 8 and 4 neurons using Dense layers with ReLU activation (similar to the first layer).

model.add(Dense(2, kernel\_initializer='normal')) adds the output layer with 2 neurons (for W and H values).

Compiles the model with:

optimizer='adam' optimizer (Adam optimizer for training).

loss='mean\_squared\_error' mean squared error loss function (suitable for regression).

metrics=['mse', 'mae'] additional metrics to track (mean squared error and mean absolute error).

10. Train the Model:

model.fit(X\_train, y\_train, batch\_size=10, epochs=150, verbose=1) trains the model on the training data.

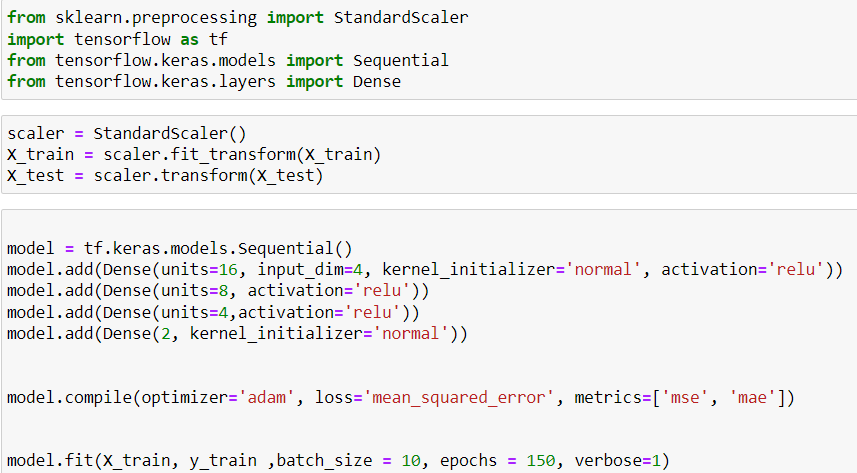
batch\_size=10 trains the model on batches of 10 samples at a time.

epochs=150 trains for 150 epochs (iterations).

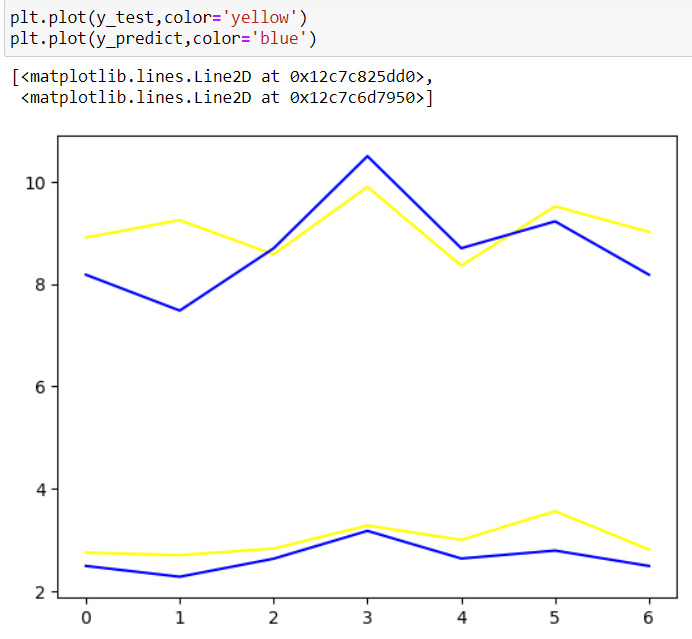
verbose=1 displays progress information during training.

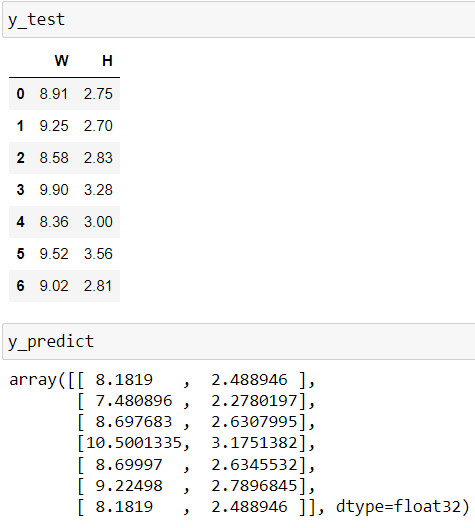
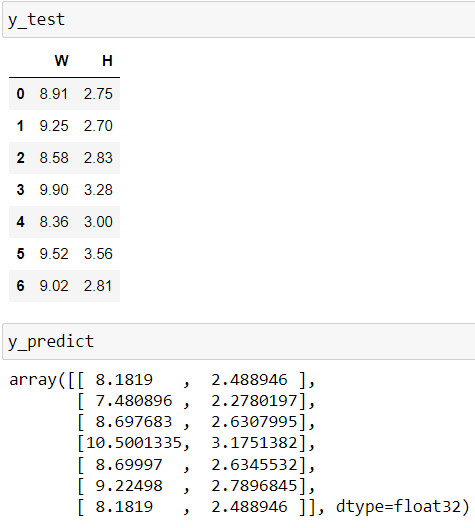
11. Make Predictions:

y\_predict = model.predict(X\_test) predicts the outputs for the testing data using the trained model.

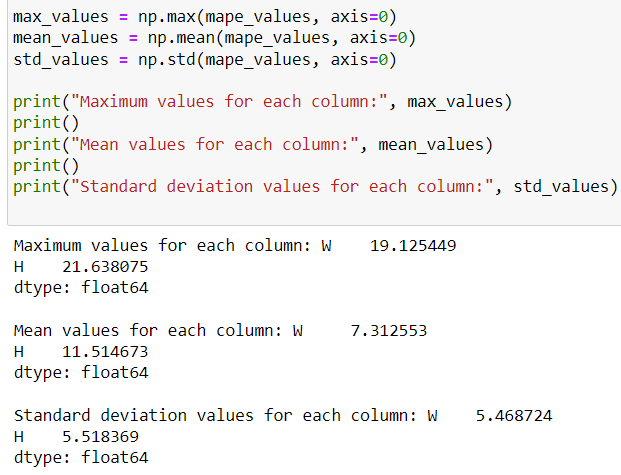
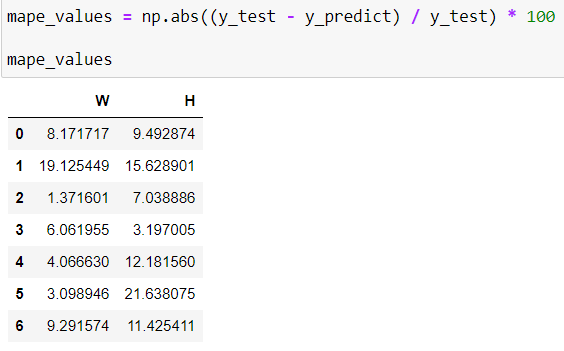


12. Plots for test and predicted values:





13.Error Analysis:



**Model 3: XGBoost**

1. Import Libraries:

numpy as np: Numerical operations.

pandas as pd: Data manipulation (reading CSV files, creating DataFrames).

from sklearn.model\_selection import train\_test\_split, GridSearchCV: Functions for data splitting and hyperparameter tuning.

from xgboost import XGBRegressor: Imports the XGBoost regressor for model building.

2. Load Data:

df = pd.read\_csv('Design\_matrix\_exp.csv', header=0) reads data from a CSV file assuming it has a header row.

3. Separate Features and Outputs:

X = df[['F', 'S', 'V', 'D']] creates a DataFrame containing the features.

y = df[['W', 'H']] creates a DataFrame containing the outputs.

4. Split Data (Train-Test):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) splits the data into training (80%) and testing (20%) sets with a random seed of 42 for reproducibility.

5. Build XGBoost Model:

xgb\_model = XGBRegressor() creates an XGBoost regressor object.

6. Define Hyperparameter Grid:

param\_grid is a dictionary containing different values to try for various hyperparameters of the XGBoost model:

n\_estimators: Number of boosting trees to create (50, 100, 150).

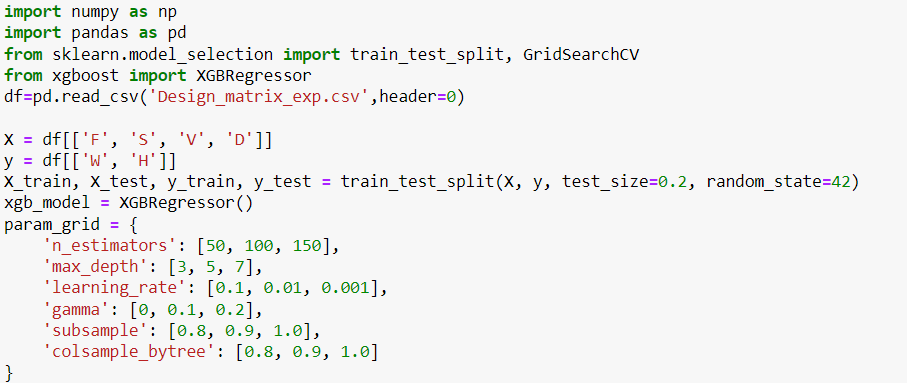
max\_depth: Maximum depth of individual trees (3, 5, 7).

learning\_rate: Learning rate for gradient boosting (0.1, 0.01, 0.001).

gamma: Minimum loss reduction required for a split to occur (0, 0.1, 0.2).

subsample: Subsample ratio of training instances (0.8, 0.9, 1.0).

colsample\_bytree: Subsample ratio of features (0.8, 0.9, 1.0).



7. Perform Grid Search with Cross-Validation:

GridSearchCV creates a grid search object to efficiently try different hyperparameter combinations.

It uses the following arguments:

estimator=xgb\_model: The XGBoost model to tune.

param\_grid: The dictionary containing hyperparameter values to try.

cv=5: Perform 5-fold cross-validation to evaluate models.

n\_jobs=-1: Use all available cores for parallel processing.

verbose=2: Print more detailed output during search.

scoring='neg\_mean\_squared\_error': Use the negative mean squared error as the scoring metric (higher negative value indicates better performance).

grid\_result = grid\_search.fit(X\_train, y\_train) performs the grid search, fitting the model with various hyperparameter combinations on the training data.

8. Print Best Parameters and Performance:

print("Best parameters found: ", grid\_result.best\_params\_) displays the best hyperparameter combination found through grid search.

print("Lowest RMSE found: ", np.sqrt(np.abs(grid\_result.best\_score\_))) calculates and prints the root mean squared error (RMSE) on the validation sets using the best hyperparameters. Lower RMSE indicates better performance.

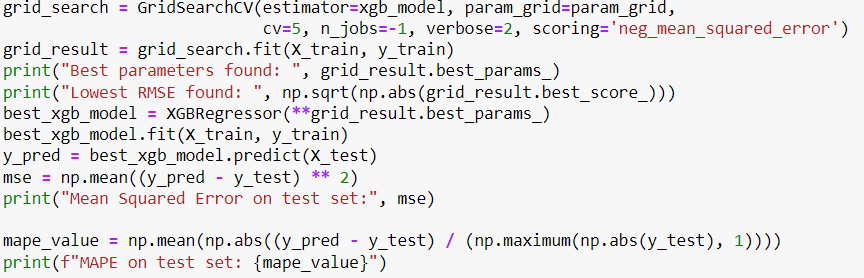
9. Train Model with Best Hyperparameters:

best\_xgb\_model = XGBRegressor(\*\*grid\_result.best\_params\_) creates a new XGBoost model with the best hyperparameters found by the grid search.

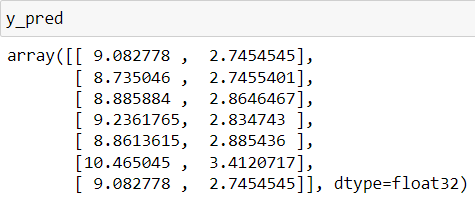
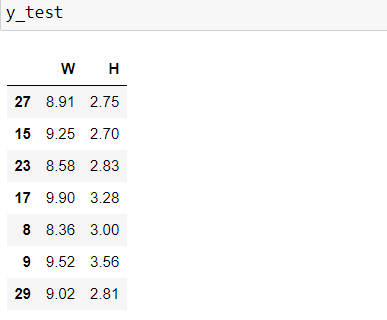
best\_xgb\_model.fit(X\_train, y\_train) trains the model with the best hyperparameters on the training data.

10. Make Predictions on Test Set:

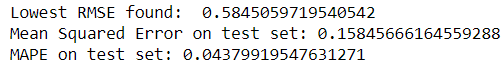
y\_pred = best\_xgb\_model.predict(X\_test) predicts the outputs for the testing data using the trained model with the best hyperparameters



11. Comparison of predicted and actual test values:



12.Error Analysis:



We continued our project and worked on the setup campus had but the parameters were changed from feed, speed, velocity and diameter to current, Wire feed speed, Torch turning speed and we also reduced the error of the Ann code.

The image of the plate on which we conducted the experiment is attached below:



The dataset obtained by performing the experiment is as follows:

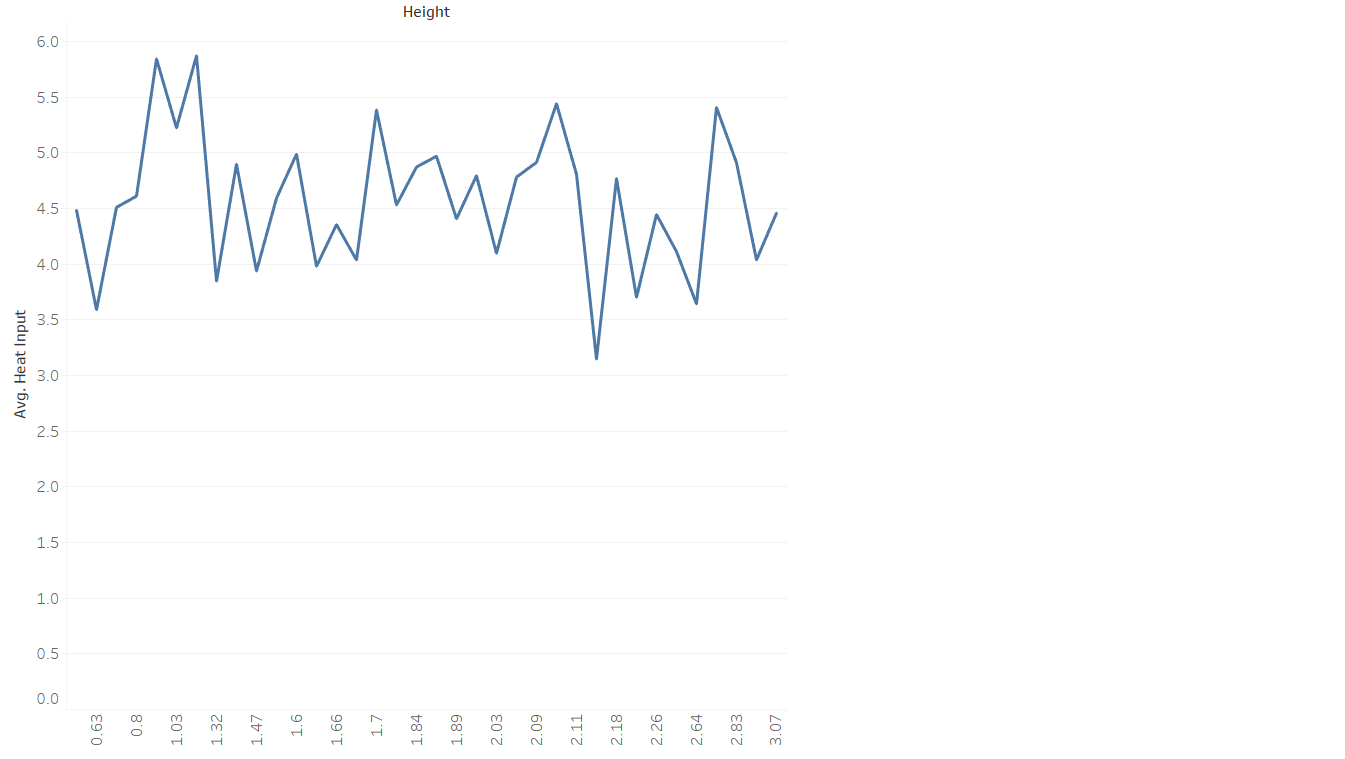
****

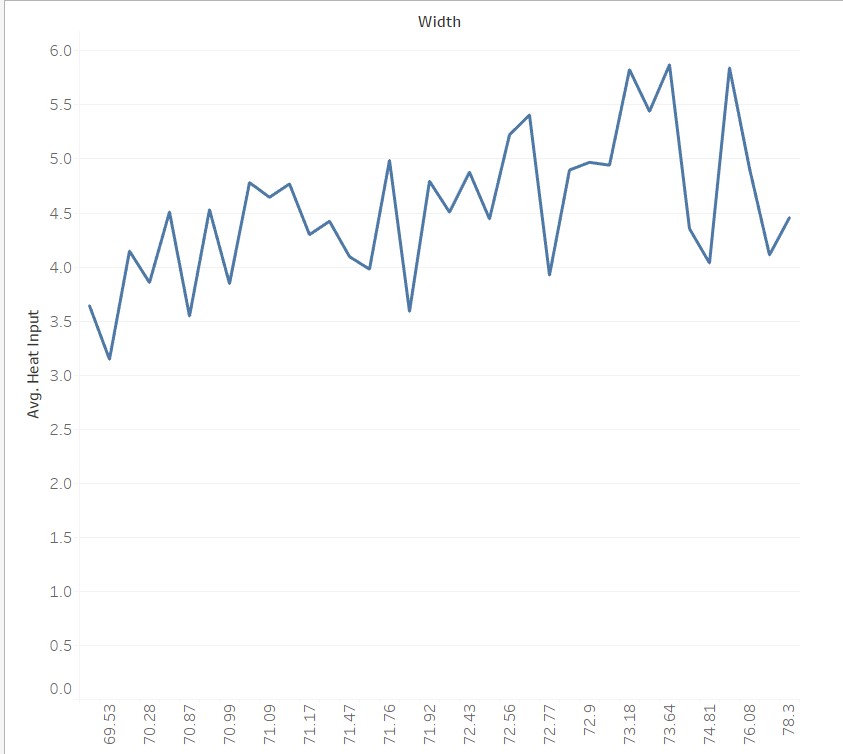
We conducted the experiment for 40 runs of which we got 3 discontinuous values which we omitted.

**Calculations:**

The average heat input is calculated as average efficiency\*voltage\*current/ tts

The graph of heat input vs height and width is as follows:





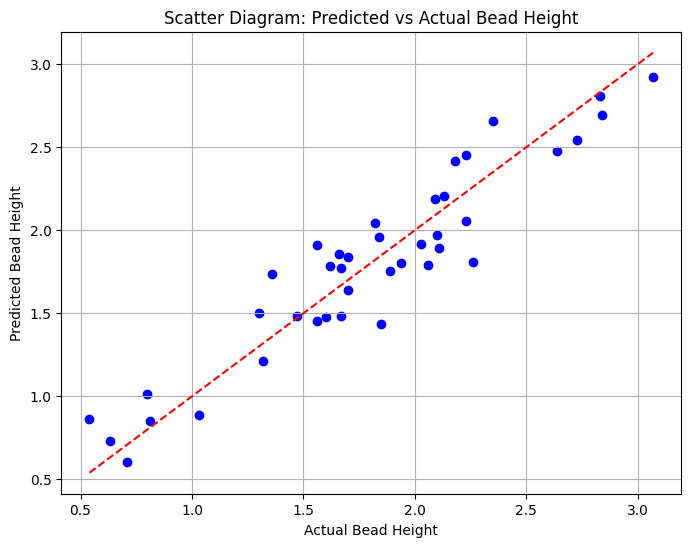
Here we can observe no trends so we can’t comment anything here.

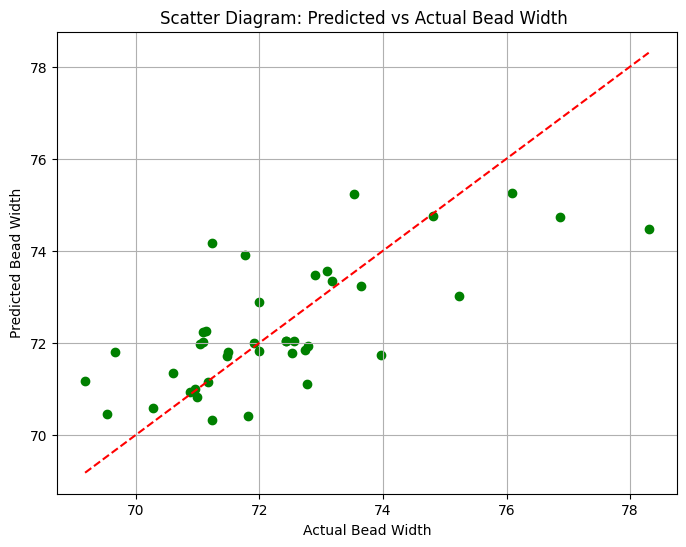
**Code:**

The code is same as mentioned above mentioned.

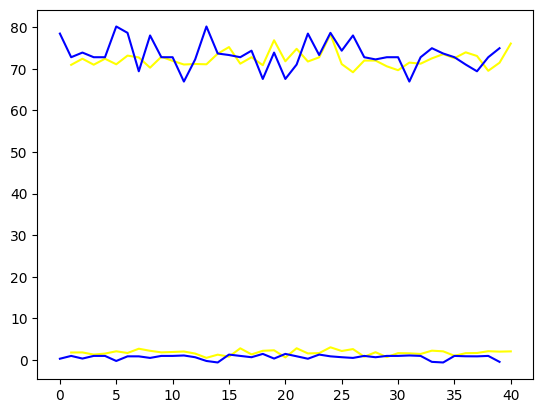
**Results:**

Graph of second order regression is as follows:





The Graph for ANN is as follows:



The **XGBoost Model** gave **MAPE** values as follows:

Mean Absolute Percentage Error for W (mm): 10%

Mean Absolute Percentage Error for H (mm): 0.9%

The **Second Order Regression** gave the **MAPE** values as follows:

Mean Absolute Percentage Error for W (mm): 11.522947360907029%

Mean Absolute Percentage Error for H (mm): 1.360699812140117%

The **ANN Model** gave **MAPE** Values as follows:

Mean Absolute Percentage Error for W (mm): 5.522947360907029%

Mean Absolute Percentage Error for H (mm): 4.2829293839393932%

**Conclusion:**

Hence, we conclude that ANN is the suitable method to predict width and SECOND ORDER REGRESSION to predict height using the parameters we used.



**REFERENCES**

1. Ahmed Yaseer, Heping Chen "Machine learning based layer roughness modeling in robotic additive manufacturing" Journal of Manufacturing Processes 70 (2021) 543–552
2. Chunyang Xia, Zengxi Pan, Joseph Polden, Huijun Li, Yanling Xu, Shanben Chen "Modelling and prediction of surface roughness in wire arc additive manufacturing using machine learning" Journal of Intelligent Manufacturing (2022) 33:1467–1482
3. Jun Xiong, Guangjun Zhang, Jianwen Hu, Lin Wu " Bead geometry prediction for robotic GMAW-based rapid manufacturing through a neural network and a second-order regression analysis" J Intell Manuf (2014) 25:157–163
4. Abderrachid Hamrani, Arvind Agarwal, Amine Allouhi, Dwayne McDaniel "Applying machine learning to wire arc additive manufacturing: a systematic data-driven literature review" Journal of Intelligent Manufacturing