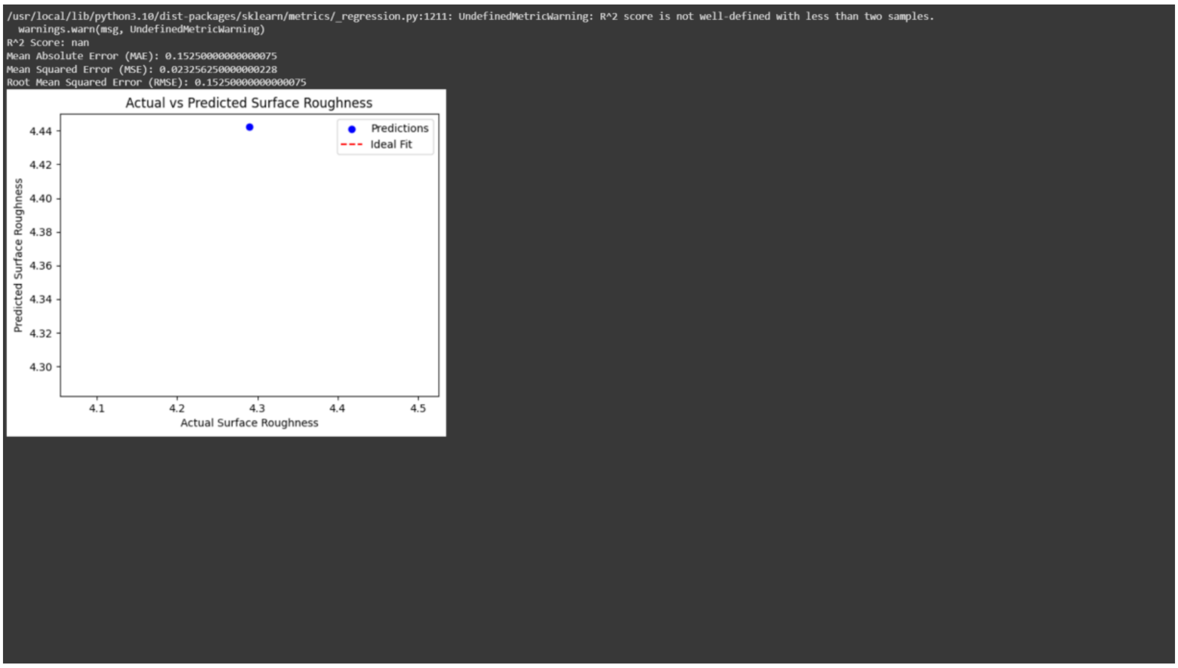
```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import math


# Example Data (Replace this with your actual dataset)

# Columns: wt% TiC, Laser Power (W), Velocity (mm/s), Gas Flow Pressure (MPa), Pulse Frequency (Hz), Surface Roughness

data = {

    'wt% TiC': [3,3,3,3,3],

    'Laser Power (W)': [2000,2000,2000,2500,2500],

    'Velocity (mm/s)': [10,10,10,20,20],

    'Gas Flow Pressure (MPa)': [0.7,0.7,0.7,1,1],

    'Pulse Frequency (Hz)': [7,10,13,7,10],

    'Surface Roughness': [4.21,4.29,4.36,4.58,4.62]

}

# Converting data to a DataFrame

df = pd.DataFrame(data)


# Splitting data into features (X) and target (y)

X = df[['wt% TiC', 'Laser Power (W)', 'Velocity (mm/s)', 'Gas Flow Pressure (MPa)', 'Pulse Frequency (Hz)']]

y = df['Surface Roughness']


# Splitting data into training and testing sets (80% training, 20% testing)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initializing the KNN Regressor

knn = KNeighborsRegressor(n_neighbors=4)  # K=5
```

```python
# Training the model
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

# Calculating performance metrics
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

# Printing results
print(f"R^2 Score: {r2}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")

# Optional: Visualize actual vs predicted values
import matplotlib.pyplot as plt

plt.scatter(y_test, y_pred, color='blue', label='Predictions')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', label='Ideal Fit')
plt.xlabel('Actual Surface Roughness')
plt.ylabel('Predicted Surface Roughness')
plt.legend()
plt.title('Actual vs Predicted Surface Roughness')
plt.show()
```




```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import math


# Example Data (Replace this with your actual dataset)

# Columns: wt% TiC, Laser Power (W), Velocity (mm/s), Gas Flow Pressure (MPa), Pulse Frequency (Hz), Surface Roughness

data = {
```

```python
    'wt% TiC': [3,3,3,3,3],
    'Laser Power (W)': [2000,2000,2000,2500,2500],
    'Velocity (mm/s)': [10,10,10,20,20],
    'Gas Flow Pressure (MPa)': [0.7,0.7,0.7,1,1],
    'Pulse Frequency (Hz)': [7,10,13,7,10],
    'Surface Roughness': [4.21,4.29,4.36,4.58,4.62]

}

# Converting data to a DataFrame
df = pd.DataFrame(data)

# Splitting data into features (X) and target (y)
X = df[['wt% TiC', 'Laser Power (W)', 'Velocity (mm/s)', 'Gas Flow Pressure (MPa)', 'Pulse Frequency (Hz)']]
y = df['Surface Roughness']

# Splitting data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scaling the data for better performance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Building the ANN model
model = Sequential([
    Dense(64, activation='relu', input_dim=X_train_scaled.shape[1]),  # Input layer
    Dense(32, activation='relu'),  # Hidden layer 1
    Dense(16, activation='relu'),  # Hidden layer 2
    Dense(1, activation='linear')  # Output layer
```

```
])

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Training the model
history = model.fit(X_train_scaled, y_train, epochs=100, batch_size=8, verbose=1,
validation_split=0.1)

# Predicting on the test set
y_pred = model.predict(X_test_scaled)

# Calculating performance metrics
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

# Printing results
print(f"R^2 Score: {r2}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")

# Optional: Visualize training history
import matplotlib.pyplot as plt

# Plot loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
```

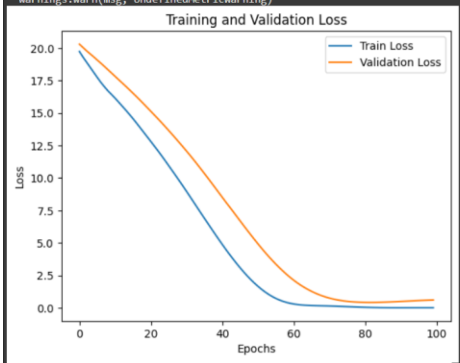
```
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()

# Plot actual vs predicted values
plt.scatter(y_test, y_pred, color='blue', label='Predictions')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--',
label='Ideal Fit')
plt.xlabel('Actual Surface Roughness')
plt.ylabel('Predicted Surface Roughness')
plt.legend()
plt.title('Actual vs Predicted Surface Roughness')
plt.show()
```

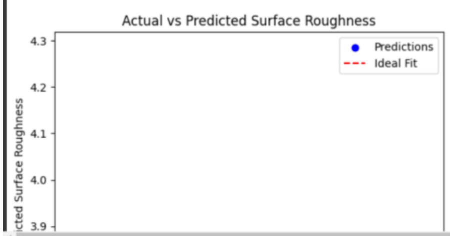
```
Epoch 1/100
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - loss: 19.7374 - mae: 4.4404 - val_loss: 20.3002 - val_mae: 4.5056
Epoch 2/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step - loss: 19.3081 - mae: 4.3918 - val_loss: 20.0452 - val_mae: 4.4772
Epoch 3/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 18.9137 - mae: 4.3465 - val_loss: 19.7902 - val_mae: 4.4486
Epoch 4/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 18.5222 - mae: 4.3010 - val_loss: 19.5560 - val_mae: 4.4222
Epoch 5/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step - loss: 18.1319 - mae: 4.2551 - val_loss: 19.3238 - val_mae: 4.3959
Epoch 6/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 17.7415 - mae: 4.2087 - val_loss: 19.0771 - val_mae: 4.3677
Epoch 7/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 17.3610 - mae: 4.1630 - val_loss: 18.8346 - val_mae: 4.3399
Epoch 8/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 17.0202 - mae: 4.1215 - val_loss: 18.5870 - val_mae: 4.3113
Epoch 9/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 16.6983 - mae: 4.0820 - val_loss: 18.3292 - val_mae: 4.2813
Epoch 10/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 16.4235 - mae: 4.0481 - val_loss: 18.0741 - val_mae: 4.2514
Epoch 11/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 16.1383 - mae: 4.0127 - val_loss: 17.8190 - val_mae: 4.2213
Epoch 12/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 15.8344 - mae: 3.9747 - val_loss: 17.5632 - val_mae: 4.1908
Epoch 13/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - loss: 15.5254 - mae: 3.9356 - val_loss: 17.3080 - val_mae: 4.1603
Epoch 14/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step - loss: 15.2143 - mae: 3.8958 - val_loss: 17.0522 - val_mae: 4.1294
Epoch 15/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step - loss: 14.8923 - mae: 3.8541 - val_loss: 16.7896 - val_mae: 4.0975
Epoch 16/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 14.5626 - mae: 3.8110 - val_loss: 16.5218 - val_mae: 4.0647
Epoch 17/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 14.2189 - mae: 3.7655 - val_loss: 16.2546 - val_mae: 4.0317
Epoch 18/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step - loss: 13.8688 - mae: 3.7185 - val_loss: 15.9802 - val_mae: 3.9975
Epoch 19/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 13.5135 - mae: 3.6702 - val_loss: 15.6998 - val_mae: 3.9623
Epoch 20/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 13.1581 - mae: 3.6214 - val_loss: 15.4137 - val_mae: 3.9260
Epoch 21/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 12.8047 - mae: 3.5722 - val_loss: 15.1269 - val_mae: 3.8893
Epoch 22/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 12.4444 - mae: 3.5212 - val_loss: 14.8392 - val_mae: 3.8522
Epoch 23/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 12.0828 - mae: 3.4693 - val_loss: 14.5501 - val_mae: 3.8145
```

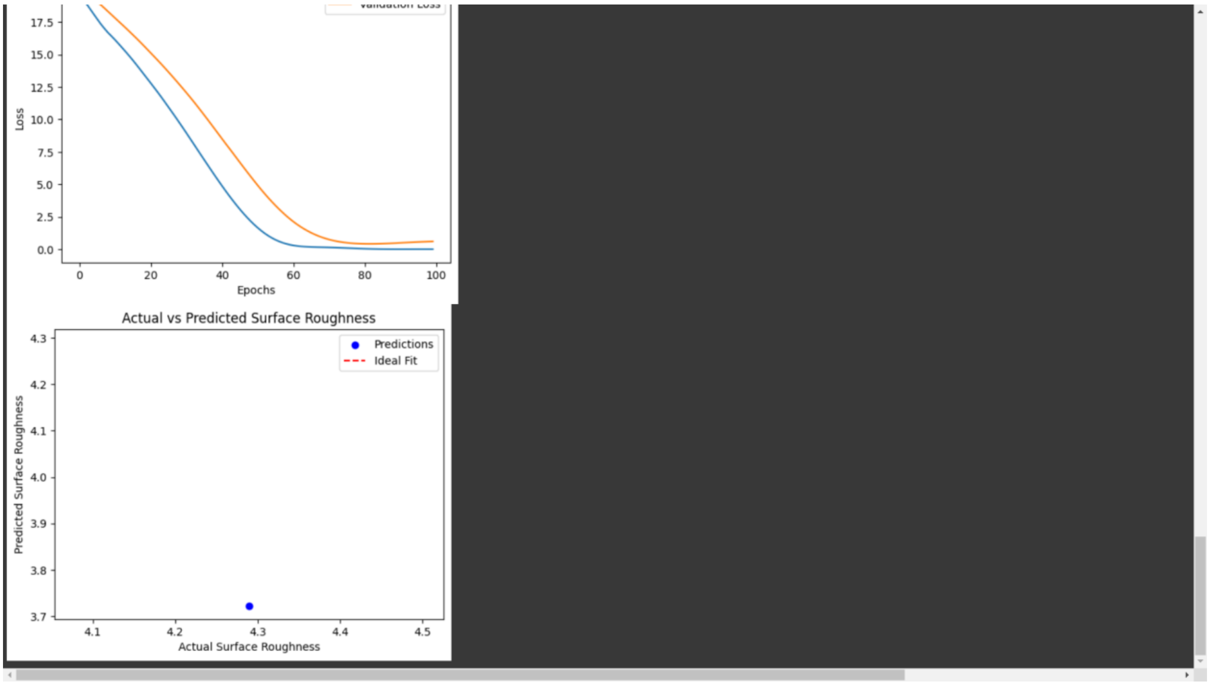
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - loss: 11.3296 - mae: 3.3586 - val_loss: 13.9552 - val_mae: 3.7357
Epoch 26/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step - loss: 10.9416 - mae: 3.3001 - val_loss: 13.6502 - val_mae: 3.6946
Epoch 27/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 10.5536 - mae: 3.2404 - val_loss: 13.3405 - val_mae: 3.6525
Epoch 28/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 10.1637 - mae: 3.1792 - val_loss: 13.0262 - val_mae: 3.6092
Epoch 29/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 9.7648 - mae: 3.1152 - val_loss: 12.7079 - val_mae: 3.5648
Epoch 30/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 9.3621 - mae: 3.0493 - val_loss: 12.3855 - val_mae: 3.5193
Epoch 31/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 8.9554 - mae: 2.9813 - val_loss: 12.0593 - val_mae: 3.4727
Epoch 32/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - loss: 8.5433 - mae: 2.9106 - val_loss: 11.7294 - val_mae: 3.4248
Epoch 33/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 8.1292 - mae: 2.8377 - val_loss: 11.3908 - val_mae: 3.3750
Epoch 34/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 7.7131 - mae: 2.7625 - val_loss: 11.0428 - val_mae: 3.3231
Epoch 35/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 7.2983 - mae: 2.6853 - val_loss: 10.6915 - val_mae: 3.2698
Epoch 36/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 6.8868 - mae: 2.6063 - val_loss: 10.3338 - val_mae: 3.2146
Epoch 37/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - loss: 6.4767 - mae: 2.5251 - val_loss: 9.9736 - val_mae: 3.1581
Epoch 38/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 6.0691 - mae: 2.4415 - val_loss: 9.6078 - val_mae: 3.0996
Epoch 39/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - loss: 5.6653 - mae: 2.3557 - val_loss: 9.2406 - val_mae: 3.0398
Epoch 40/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 5.2668 - mae: 2.2678 - val_loss: 8.8725 - val_mae: 2.9787
Epoch 41/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - loss: 4.8750 - mae: 2.1776 - val_loss: 8.5041 - val_mae: 2.9162
Epoch 42/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - loss: 4.4917 - mae: 2.0855 - val_loss: 8.1362 - val_mae: 2.8524
Epoch 43/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 4.1182 - mae: 1.9915 - val_loss: 7.7795 - val_mae: 2.7892
Epoch 44/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - loss: 3.7572 - mae: 1.8962 - val_loss: 7.4148 - val_mae: 2.7230
Epoch 45/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 3.4081 - mae: 1.7991 - val_loss: 7.0486 - val_mae: 2.6549
Epoch 46/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 3.0740 - mae: 1.7010 - val_loss: 6.6834 - val_mae: 2.5852
Epoch 47/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 2.7541 - mae: 1.6012 - val_loss: 6.3203 - val_mae: 2.5140
Epoch 48/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - loss: 2.4508 - mae: 1.5005 - val_loss: 5.9613 - val_mae: 2.4416
Epoch 49/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step - loss: 2.1650 - mae: 1.3988 - val_loss: 5.6070 - val_mae: 2.3679
```

```
Epoch 75/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step - loss: 0.1059 - mae: 0.2810 - val_loss: 0.5341 - val_mae: 0.7308
Epoch 76/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 0.0946 - mae: 0.2762 - val_loss: 0.5009 - val_mae: 0.7077
Epoch 77/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.0832 - mae: 0.2676 - val_loss: 0.4742 - val_mae: 0.6887
Epoch 78/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.0721 - mae: 0.2558 - val_loss: 0.4536 - val_mae: 0.6735
Epoch 79/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.0614 - mae: 0.2411 - val_loss: 0.4383 - val_mae: 0.6621
Epoch 80/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - loss: 0.0514 - mae: 0.2241 - val_loss: 0.4276 - val_mae: 0.6539
Epoch 81/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0425 - mae: 0.2057 - val_loss: 0.4211 - val_mae: 0.6489
Epoch 82/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step - loss: 0.0347 - mae: 0.1860 - val_loss: 0.4183 - val_mae: 0.6467
Epoch 83/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.0278 - mae: 0.1654 - val_loss: 0.4188 - val_mae: 0.6471
Epoch 84/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step - loss: 0.0221 - mae: 0.1442 - val_loss: 0.4222 - val_mae: 0.6498
Epoch 85/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.0175 - mae: 0.1228 - val_loss: 0.4281 - val_mae: 0.6543
Epoch 86/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 0.0140 - mae: 0.1016 - val_loss: 0.4361 - val_mae: 0.6604
Epoch 87/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - loss: 0.0113 - mae: 0.0809 - val_loss: 0.4460 - val_mae: 0.6678
Epoch 88/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step - loss: 0.0096 - mae: 0.0665 - val_loss: 0.4573 - val_mae: 0.6762
Epoch 89/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.0085 - mae: 0.0634 - val_loss: 0.4697 - val_mae: 0.6854
Epoch 90/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.0080 - mae: 0.0721 - val_loss: 0.4830 - val_mae: 0.6950
Epoch 91/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.0080 - mae: 0.0801 - val_loss: 0.4969 - val_mae: 0.7049
Epoch 92/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step - loss: 0.0083 - mae: 0.0869 - val_loss: 0.5108 - val_mae: 0.7147
Epoch 93/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.0089 - mae: 0.0926 - val_loss: 0.5246 - val_mae: 0.7243
Epoch 94/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.0096 - mae: 0.0967 - val_loss: 0.5381 - val_mae: 0.7335
Epoch 95/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.0101 - mae: 0.0994 - val_loss: 0.5512 - val_mae: 0.7424
Epoch 96/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.0106 - mae: 0.1006 - val_loss: 0.5638 - val_mae: 0.7508
Epoch 97/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 0.0108 - mae: 0.1006 - val_loss: 0.5757 - val_mae: 0.7588
Epoch 98/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.0108 - mae: 0.0994 - val_loss: 0.5869 - val_mae: 0.7661
```

```
R^2 Score: nan
Mean Absolute Error (MAE): 0.5676234149932862
Mean Squared Error (MSE): 0.3221963412486404
Root Mean Squared Error (RMSE): 0.5676234149932862
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:1211: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```








```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.neighbors import KNeighborsRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import math
import matplotlib.pyplot as plt

# Example Data (Replace this with your actual dataset)
data = {
```

```
    'wt% TiC': [3,3,3,3,3],
    'Laser Power (W)': [2000,2000,2000,2500,2500],
    'Velocity (mm/s)': [10,10,10,20,20],
    'Gas Flow Pressure (MPa)': [0.7,0.7,0.7,1,1],
    'Pulse Frequency (Hz)': [7,10,13,7,10],
    'Surface Roughness': [4.21,4.29,4.36,4.58,4.62]
}

# Converting data to a DataFrame
df = pd.DataFrame(data)

# Splitting data into features (X) and target (y)
X = df[['wt% TiC', 'Laser Power (W)', 'Velocity (mm/s)', 'Gas Flow Pressure (MPa)', 'Pulse Frequency (Hz)']]
y = df['Surface Roughness']

# Splitting data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scaling the data for ANN and better performance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ---- KNN Model ----
knn = KNeighborsRegressor(n_neighbors=4)
knn.fit(X_train, y_train)
knn_predictions = knn.predict(X_test)

# ---- ANN Model ----
# Building the ANN
```

```
model = Sequential([
    Dense(64, activation='relu', input_dim=X_train_scaled.shape[1]),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1, activation='linear')
])

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
model.fit(X_train_scaled, y_train, epochs=100, batch_size=8, verbose=1, validation_split=0.1)

# ANN Predictions
ann_predictions = model.predict(X_test_scaled).flatten()

# ---- Combining Predictions ----
# Simple Averaging
ensemble_predictions = (knn_predictions + ann_predictions) / 2

# ---- Performance Metrics ----
def calculate_metrics(y_true, y_pred, model_name):
    r2 = r2_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = math.sqrt(mse)
    print(f"\n{model_name} Performance:")
    print(f"R^2 Score: {r2}")
    print(f"Mean Absolute Error (MAE): {mae}")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"Root Mean Squared Error (RMSE): {rmse}")
    return r2, mae, mse, rmse

# Metrics for KNN
```

```
calculate_metrics(y_test, knn_predictions, "KNN")

# Metrics for ANN
calculate_metrics(y_test, ann_predictions, "ANN")

# Metrics for Ensemble
calculate_metrics(y_test, ensemble_predictions, "Ensemble (KNN + ANN)")

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()

# ---- Visualization ----
plt.scatter(y_test, knn_predictions, color='blue', label='KNN Predictions')
plt.scatter(y_test, ann_predictions, color='green', label='ANN Predictions')
plt.scatter(y_test, ensemble_predictions, color='purple', label='Ensemble Predictions')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--',
label='Ideal Fit')
plt.xlabel('Actual Surface Roughness')
plt.ylabel('Predicted Surface Roughness')
plt.legend()
plt.title('Actual vs Predicted Surface Roughness')
```

plt.show()

```
KNN Performance:
R^2 Score: nan
Mean Absolute Error (MAE): 0.15250000000000075
Mean Squared Error (MSE): 0.023256250000000228
Root Mean Squared Error (RMSE): 0.15250000000000075

ANN Performance:
R^2 Score: nan
Mean Absolute Error (MAE): 0.1926432037353516
Mean Squared Error (MSE): 0.03711140394542018
Root Mean Squared Error (RMSE): 0.1926432037353516

Ensemble (KNN + ANN) Performance:
R^2 Score: nan
Mean Absolute Error (MAE): 0.020071601867675426
Mean Squared Error (MSE): 0.00040286920153447165
Root Mean Squared Error (RMSE): 0.020071601867675426
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:1211: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:1211: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:1211: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```