Computational Prediction of Critical Temperatures in Superconducting Materials using Deep Bottleneck Neural Networks

Dataset: UCI Superconductivity Dataset

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Team Members:
Nilay Singh 102313043

Submitted to :
Dr Jasmeet Singh

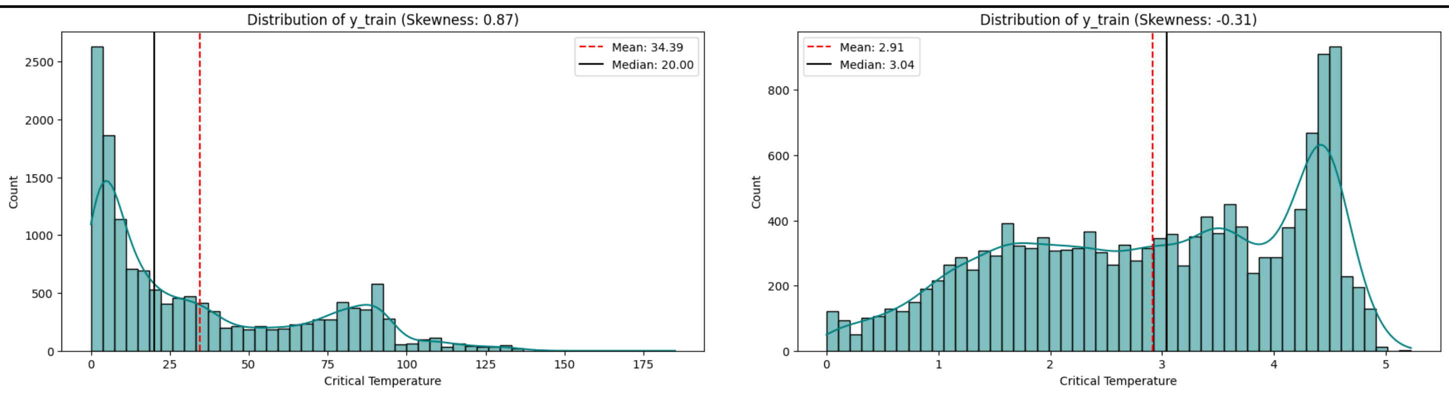
1. Introduction

Identifying the critical temperature (Tc) at which a material exhibits superconductivity is a fundamental challenge in material science. This project applies a Deep Learning approach to predict Tc using 81 chemical and physical properties. By automating this prediction, we provide a high-speed alternative to traditional laboratory synthesis and testing.

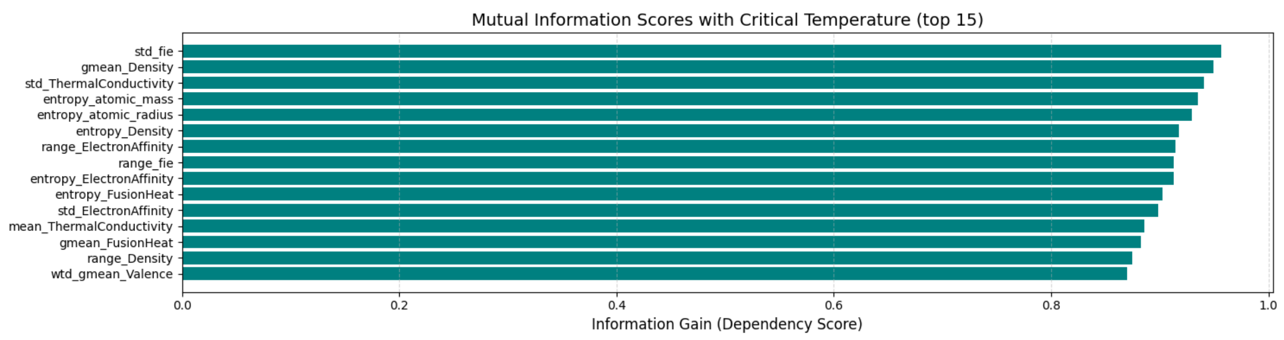
2. Data Collection and Pre-processing

The dataset consists of 21,263 materials. Successful training required a robust stages preprocessing pipeline:

- **Target Normalization:** The raw Tc values exhibited a heavy right-skew (0.87). We applied a **Log(1+x) transformation**, which shifted the skewness to **-0.31**. This "Gaussian-like" distribution was critical for achieving our high R2 score, as it allowed the model to treat errors at low and high temperatures with equal mathematical weight.
- **Feature Standardization:** We utilized **StandardScaler** on the 81 input features. This ensured that features with large numerical ranges (e.g., atomic mass) did not dominate the weights during the gradient descent process.
- The Dataset was partitioned into 3 parts - **Train(64%),Validate(16%) and Test Set(20%)**
- Before training, we conducted a **Mutual Information (MI)** analysis to quantify the statistical dependence between the 81 physical features and the critical temperature Tc and to perform Feature Selection if necessary.



As shown in the histograms, the transformation corrected the initial skewness. This shift to a more normal distribution is why the model achieved such high stability and predictive power (R2= 0.8791).



The MI analysis revealed consistently high scores across nearly all 81 features, from atomic radius to thermal conductivity. This indicates that predictive signals are distributed throughout the physical properties rather than concentrated in a few variables.

Since the scores were robust across the entire set, we bypassed feature selection. Training on the full 81-dimensional dataset allowed the Deep Learning architecture to internally prioritize features and capture the complex, multi-variable interactions essential for predicting superconductivity.

4. Deep Learning Model Description

We implemented a **High-Capacity Bottleneck Neural Network** using the Keras Sequential API. The model expands the 81 input features into higher-dimensional representations and then compresses them through a bottleneck for accurate regression.

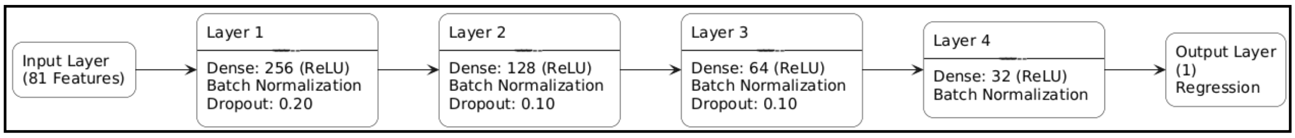
Architecture: 81 → 256 → 128 → 64 → 32 → 1

Activations & Normalization: ReLU is applied to all hidden layers, and Batch Normalization follows each dense layer to stabilize training and accelerate convergence.

Regularization: A progressive Dropout strategy (starting at 0.2 and decreasing in deeper layers) was used to prevent overfitting.

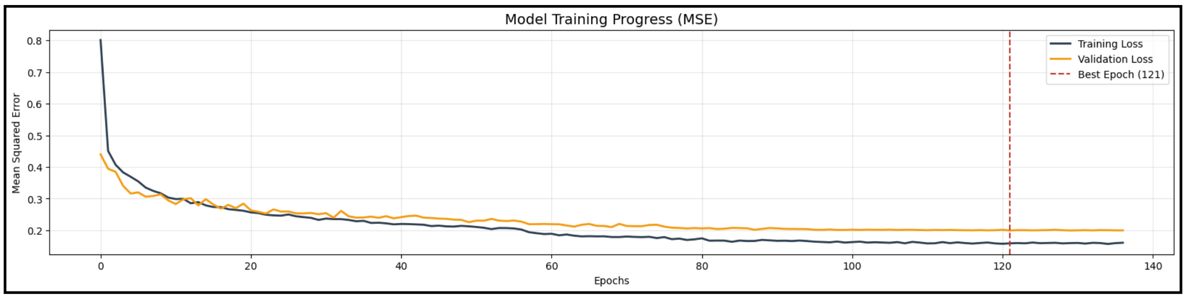
Optimization: Training used the Adam optimizer with:

- **ReduceLROnPlateau** to lower learning rate when validation loss stagnated.
- **Early Stopping** (patience = 20) to restore the best weights and prevent overfitting.



5. Training Performance

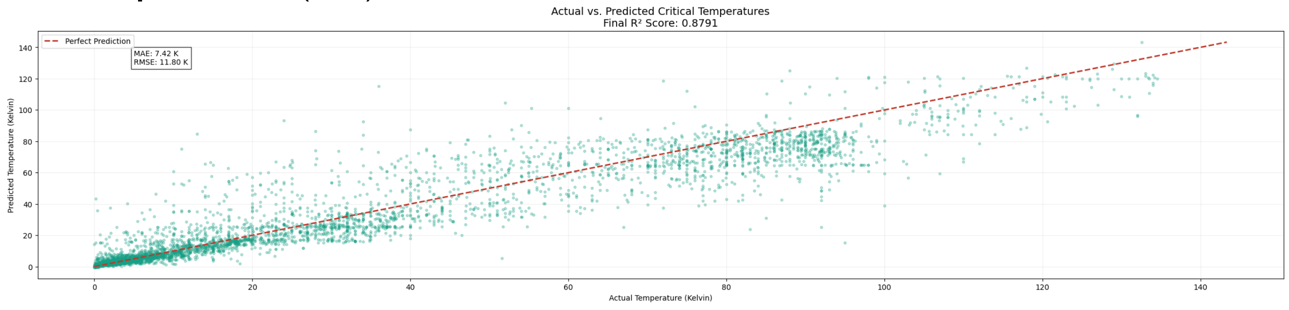
The training history demonstrates a smooth convergence between the training and validation sets.



6. Final Results and Discussion

The model was evaluated on the 20% hold-out test set. All predictions were inverse-transformed from the log-scale back to Kelvin to measure real-world physical accuracy.

- **Mean Absolute Error (MAE):** 7.42 K
- **Root Mean Squared Error (RMSE):** 11.80 K
- **R-squared Score (R^2):** 0.8791



Graph Explanation: The Actual vs. Predicted Plot shows a high density of data points clustered tightly along the 45-degree identity line. The R2 of 0.8791 indicates that the model explains nearly 88% of the variance in critical temperatures. The slight variance observed at higher temperatures (>80 K) accounts for the RMSE being higher than the MAE, reflecting the inherent complexity of high-Tc superconductors.

7. Conclusion

The project successfully developed a robust Deep Learning pipeline for material science. With an average error of only 7.42 K, this model serves as an effective computational tool for narrowing down candidate materials for experimental superconductivity research.

8. Code

a. Importing and Preprocessing

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from ucimlrepo import fetch_ucirepo
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 import layers, models, callbacks


superconductivty_data = fetch_ucirepo(id=464)


X = superconductivty_data.data.features
y = superconductivty_data.data.targets


X_train_full, X_test, y_train_full, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.2, random_state=42
)


y_train_log = np.log1p(y_train.values.flatten())
y_val_log = np.log1p(y_val.values.flatten())


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

b. Feature Importance and Mutual Information code

```python
from sklearn.feature_selection import mutual_info_regression

def perform_mi_analysis(X_scaled, y_log, feature_names):
    print("Calculating Mutual Information scores...")
    mi_scores = mutual_info_regression(X_scaled, y_log, random_state=42)

    mi_series = pd.Series(mi_scores, index=feature_names)
    mi_series = mi_series.sort_values(ascending=False)
    plt.figure(figsize=(10, 8))
    sns.barplot(x=mi_series.values[:20], y=mi_series.index[:20], palette='magma')

    plt.title("Top 20 Features by Mutual Information Score", fontsize=14, fontweight='bold')
    plt.xlabel("Mutual Information Score (Information Gain)")
    plt.ylabel("Physical Property")
    plt.grid(axis='x', linestyle='--', alpha=0.4)
    plt.tight_layout()
    plt.show()

    return mi_series

mi_results = perform_mi_analysis(X_train_scaled, y_train_log, X.columns)
```

b. Model Training and Evaluation

```
def build_final_model():
    model = models.Sequential([

        layers.Dense(256, activation='relu', input_shape=(X_train_scaled.shape[1],)),
        layers.BatchNormalization(),
        layers.Dropout(0.2),


        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.1),

        layers.Dense(64, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.1),

        layers.Dense(32, activation='relu'),


        layers.Dense(1)
    ])

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
        loss='mse',
        metrics=['mae']
    )
    return model

model = build_final_model()

reduce_lr = callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=7)
early_stop = callbacks.EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)

history = model.fit(
    X_train_scaled, y_train_log,
    validation_data=(X_val_scaled, y_val_log),
    epochs=200, batch_size=16,
    callbacks=[reduce_lr, early_stop],
    verbose=0
)
log_predictions = model1.predict(X_test_scaled)
final_predictions = np.expm1(log_predictions).flatten()

mae = mean_absolute_error(y_test, final_predictions)
rmse = np.sqrt(mean_squared_error(y_test, final_predictions))
r2 = r2_score(y_test, final_predictions)

print(f"--- Test Set Results ---")
print(f"Mean Absolute Error (MAE): {mae:.2f} K")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f} K")
print(f"R-squared Score (R2): {r2:.4f}")
```