## Step Two: Model Training

The next step is training the C2P(Composition to Performance) and P2C(Performance to Composition).

It will include feeding the previously per-processed data into the 2 models multiple times and monitoring the predictions by the model then using a loss function and optimizer to minimize the error in prediction

and hence improving performance.

#### **Model architectures**:

##### C2P:

The C2P model is a feed forward neural network to be implemented using TensorFlow and/or Keras. It has of three layers:

* Input layer: Takes the normalized input features (composition and heat treatment parameters).
* Hidden layers: 2 layers with ReLU activation function, consisting of 64 and 32 neurons respectively.
* Output layer: A layer with two neurons and a linear activation function, corresponding to the predicted tensile strength and elongation.

Compilation will involve defining the optimizer, loss function, and metrics that the model will use to learn and evaluate its performance.

During compilation stage, the Adam optimizer in Keras will be used to minimize the loss function during training.

The loss function to be used is MSE(Mean Squared Error), to give differences between the prediction and actual values. It will ensure larger errors are appear more significant than smaller ones, for accuracy.

MAE(Mean Absolute Error) will also be used for more easy to interpret values.

##### P2C:

The P2C model is also a feed forward neural network, similar in structure to the C2P model but reversed:

* Input layer:It takes the desired mechanical properties (tensile strength and elongation).
* Hidden layers: also 2 layers with ReLU activation function, consisting of 64 and 32 neurons respectively.
* Output layer: A layer with the same number of neurons as the input features, corresponding to the predicted composition and heat treatment parameters

The same compiler and loss functions as the C2P model will also apply here for compilation.

#### Code for C2P:

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.metrics import mean\_absolute\_percentage\_error

# Load the data from a CSV file

data = pd.read\_csv('spring\_steel\_data.csv')

# Separate the features (X) and target variables (y)

features = data[['C', 'Si', 'Mn', 'Cr', 'V', 'quench\_temp', 'quench\_time', 'temper\_temp', 'temper\_time']]

target = data[['tensile\_strength', 'elongation']]

# Split the data into training and test sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.2, random\_state=42)

# Normalize the features using StandardScaler

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Function to create the C2P model

def create\_c2p\_model(input\_dim):

    model = Sequential([

        Dense(64, input\_dim=input\_dim, activation='relu'),  # First hidden layer with 64 neurons

        Dense(32, activation='relu'),                      # Second hidden layer with 32 neurons

        Dense(2, activation='linear')                      # Output layer with 2 neurons (for tensile\_strength and elongation)

    ])

    # Compile the model with Adam optimizer and Mean Squared Error loss function

    model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=['mae'])

    return model

# Create the model with the number of input features

c2p\_model = create\_c2p\_model(X\_train\_scaled.shape[1])

# Train the C2P model

c2p\_model.fit(X\_train\_scaled, y\_train, epochs=100, batch\_size=32, validation\_split=0.2)

# Predict on the test set

y\_pred = c2p\_model.predict(X\_test\_scaled)

# Calculate Mean Absolute Percentage Error (MAPE)

mape = mean\_absolute\_percentage\_error(y\_test, y\_pred)

print(f'C2P Model MAPE: {mape}')