resample_length

July 3, 2025

[1]:
```
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping, ModelCheckpoint
#from keras_flops import get_flops
import matplotlib.pyplot as plt
import joblib

# 1. Data Loading and Preparation
def load_time_series_data():
    """Load data where each test has one time series of temperatures"""
    summary_df = pd.read_csv('all_time.csv')
    test_data = []

    for i in range(1, 26):
        try:
            filename = f'csv_ave/{i}.csv'
            df = pd.read_csv(filename, header=1)
            test_params = summary_df.iloc[i-1][['Fb_set\n[kN]', 'v_set\n[km/
 ↪h]', ' _m', 't2\n[s]']].values
            temperature_series = df.iloc[:, 3].values

            test_data.append({
                'Fb_set': test_params[0],
                'v_set': test_params[1],
                ' _m': test_params[2],
                't2': test_params[3],
                'temperature_series': temperature_series
            })
        except FileNotFoundError:
            continue
```

```
    return test_data

def resample_series(temp_series, original_length, target_length=1500):
    """Resample time series to target length using linear interpolation"""
    original_time = np.linspace(0, 1, original_length)
    target_time = np.linspace(0, 1, target_length)
    return np.interp(target_time, original_time, temp_series)

# Load and preprocess data
test_data = load_time_series_data()
TARGET_LENGTH = 500  # Fixed length for all series

X = []
y = []

for test in test_data:
    # Input features
    X.append([test['Fb_set'], test['v_set'], test[' _m'], test['t2']])

    # Resample temperature series
    original_length = len(test['temperature_series'])
    resampled = resample_series(
        test['temperature_series'],
        original_length,
        TARGET_LENGTH
    )
    y.append(resampled)

X = np.array(X)
y = np.array(y)
```

[2]:
```
print(X.shape)
print(y.shape)
```

```
(11, 4)
(11, 500)
```

[3]:
```
# Train-test split
#X_train, X_test, y_train, y_test = train_test_split(
#    X, y, test_size=0.1, random_state=42

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1,random_state=42)

# Normalization
scaler_X = StandardScaler()
X_train = scaler_X.fit_transform(X_train)
X_test = scaler_X.transform(X_test)
```

```
scaler_y = StandardScaler()
y_train = scaler_y.fit_transform(y_train)
y_test = scaler_y.transform(y_test)

# 2. Model Definition
def build_model():
    inputs = keras.Input(shape=(4,))  # 4 input features

    # Feature extraction
    params = layers.Dense(64, activation='relu')(inputs)
    params = layers.Dropout(0.2)(params)

    # Repeat for time steps
    repeated = layers.RepeatVector(TARGET_LENGTH)(params)

    # Temporal processing
    lstm_out = layers.LSTM(64, return_sequences=True)(repeated)
    outputs = layers.TimeDistributed(layers.Dense(1))(lstm_out)

    model = keras.Model(inputs, outputs)
    model.compile(
        loss='mse',
        optimizer=keras.optimizers.Adam(learning_rate=0.01),
        metrics=['mae']
    )
    return model

model = build_model()
model.summary()

# 3. Training Configuration
callbacks = [
    EarlyStopping(patience=100, restore_best_weights=True),
    ModelCheckpoint('best_model.h5', save_best_only=True)
]

# 4. Training
history = model.fit(
    X_train, y_train,
    validation_split=0.1,
    epochs=50,
    batch_size=16,
    callbacks=callbacks,
    verbose=1)

# 5. Evaluation
```

```python
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=0)
print(f"Test MSE: {test_loss:.4f}, Test MAE: {test_mae:.4f}")

# 6. Save artifacts
model.save('brake_temp_model.keras')
joblib.dump(scaler_X, 'scaler_X.pkl')
joblib.dump(scaler_y, 'scaler_y.pkl')
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 4)	0
dense (Dense)	(None, 64)	320
dropout (Dropout)	(None, 64)	0
repeat_vector (RepeatVector)	(None, 500, 64)	0
lstm (LSTM)	(None, 500, 64)	33,024
time_distributed (TimeDistributed)	(None, 500, 1)	65

Total params: 33,409 (130.50 KB)

Trainable params: 33,409 (130.50 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
1/1             0s 4s/step - loss:
0.8031 - mae: 0.7756
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
1/1             5s 5s/step - loss:
0.8031 - mae: 0.7756 - val_loss: 0.8185 - val_mae: 0.8251
Epoch 2/50
```

```
1/1             0s 91ms/step - loss:
0.5712 - mae: 0.6240

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

1/1             0s 117ms/step - loss:
0.5712 - mae: 0.6240 - val_loss: 0.1669 - val_mae: 0.2490
Epoch 3/50
1/1             0s 107ms/step - loss:
0.1656 - mae: 0.2904 - val_loss: 0.2712 - val_mae: 0.4583
Epoch 4/50
1/1             0s 104ms/step - loss:
0.2213 - mae: 0.3638 - val_loss: 0.2762 - val_mae: 0.4641
Epoch 5/50
1/1             0s 104ms/step - loss:
0.1838 - mae: 0.3242 - val_loss: 0.2213 - val_mae: 0.3913
Epoch 6/50
1/1             0s 108ms/step - loss:
0.1481 - mae: 0.2756 - val_loss: 0.1774 - val_mae: 0.2944
Epoch 7/50
1/1             0s 104ms/step - loss:
0.1207 - mae: 0.2106 - val_loss: 0.1762 - val_mae: 0.2174
Epoch 8/50
1/1             0s 104ms/step - loss:
0.1382 - mae: 0.2180 - val_loss: 0.2080 - val_mae: 0.2066
Epoch 9/50
1/1             0s 104ms/step - loss:
0.1329 - mae: 0.2152 - val_loss: 0.2365 - val_mae: 0.2599
Epoch 10/50
1/1             0s 103ms/step - loss:
0.1483 - mae: 0.2402 - val_loss: 0.2165 - val_mae: 0.2197
Epoch 11/50
1/1             0s 103ms/step - loss:
0.1295 - mae: 0.2087 - val_loss: 0.1948 - val_mae: 0.1856
Epoch 12/50
1/1             0s 103ms/step - loss:
0.1369 - mae: 0.2258 - val_loss: 0.1765 - val_mae: 0.2245
Epoch 13/50
1/1             0s 104ms/step - loss:
0.1121 - mae: 0.1623 - val_loss: 0.1743 - val_mae: 0.2631
Epoch 14/50
1/1             0s 103ms/step - loss:
0.1233 - mae: 0.2114 - val_loss: 0.1777 - val_mae: 0.2878
Epoch 15/50
1/1             0s 103ms/step - loss:
```

```
0.1302 - mae: 0.2164 - val_loss: 0.1790 - val_mae: 0.2934
Epoch 16/50
1/1             0s 133ms/step - loss:
0.1562 - mae: 0.2656 - val_loss: 0.1770 - val_mae: 0.2827
Epoch 17/50
1/1             0s 108ms/step - loss:
0.1204 - mae: 0.2192 - val_loss: 0.1752 - val_mae: 0.2659
Epoch 18/50
1/1             0s 104ms/step - loss:
0.1146 - mae: 0.1881 - val_loss: 0.1751 - val_mae: 0.2455
Epoch 19/50
1/1             0s 104ms/step - loss:
0.1134 - mae: 0.1849 - val_loss: 0.1766 - val_mae: 0.2323
Epoch 20/50
1/1             0s 105ms/step - loss:
0.1403 - mae: 0.2369 - val_loss: 0.1819 - val_mae: 0.2106
Epoch 21/50
1/1             0s 103ms/step - loss:
0.1237 - mae: 0.2133 - val_loss: 0.1873 - val_mae: 0.1972
Epoch 22/50
1/1             0s 105ms/step - loss:
0.1322 - mae: 0.2356 - val_loss: 0.1885 - val_mae: 0.1948
Epoch 23/50
1/1             0s 104ms/step - loss:
0.1267 - mae: 0.2215 - val_loss: 0.1846 - val_mae: 0.2043
Epoch 24/50
1/1             0s 110ms/step - loss:
0.1297 - mae: 0.2264 - val_loss: 0.1811 - val_mae: 0.2148
Epoch 25/50
1/1             0s 104ms/step - loss:
0.1129 - mae: 0.1773 - val_loss: 0.1790 - val_mae: 0.2234
Epoch 26/50
1/1             0s 105ms/step - loss:
0.1122 - mae: 0.1849 - val_loss: 0.1769 - val_mae: 0.2353
Epoch 27/50
1/1             0s 104ms/step - loss:
0.1529 - mae: 0.2351 - val_loss: 0.1760 - val_mae: 0.2449
Epoch 28/50
1/1             0s 103ms/step - loss:
0.1180 - mae: 0.1828 - val_loss: 0.1757 - val_mae: 0.2547
Epoch 29/50
1/1             0s 103ms/step - loss:
0.1173 - mae: 0.1987 - val_loss: 0.1758 - val_mae: 0.2599
Epoch 30/50
1/1             0s 104ms/step - loss:
0.1186 - mae: 0.1867 - val_loss: 0.1762 - val_mae: 0.2664
Epoch 31/50
1/1             0s 103ms/step - loss:
```

```
0.1252 - mae: 0.2193 - val_loss: 0.1759 - val_mae: 0.2616
Epoch 32/50
1/1             0s 102ms/step - loss:
0.1222 - mae: 0.2089 - val_loss: 0.1759 - val_mae: 0.2506
Epoch 33/50
1/1             0s 103ms/step - loss:
0.1197 - mae: 0.1784 - val_loss: 0.1770 - val_mae: 0.2365
Epoch 34/50
1/1             0s 103ms/step - loss:
0.1184 - mae: 0.1993 - val_loss: 0.1809 - val_mae: 0.2175
Epoch 35/50
1/1             0s 104ms/step - loss:
0.1223 - mae: 0.2032 - val_loss: 0.1877 - val_mae: 0.1986
Epoch 36/50
1/1             0s 103ms/step - loss:
0.1118 - mae: 0.1698 - val_loss: 0.1962 - val_mae: 0.1845
Epoch 37/50
1/1             0s 106ms/step - loss:
0.1178 - mae: 0.1918 - val_loss: 0.2033 - val_mae: 0.1935
Epoch 38/50
1/1             0s 104ms/step - loss:
0.1272 - mae: 0.2360 - val_loss: 0.2016 - val_mae: 0.1909
Epoch 39/50
1/1             0s 107ms/step - loss:
0.1334 - mae: 0.2339 - val_loss: 0.1947 - val_mae: 0.1855
Epoch 40/50
1/1             0s 107ms/step - loss:
0.1251 - mae: 0.2249 - val_loss: 0.1858 - val_mae: 0.2031
Epoch 41/50
1/1             0s 132ms/step - loss:
0.1118 - mae: 0.1871 - val_loss: 0.1793 - val_mae: 0.2245
Epoch 42/50
1/1             0s 103ms/step - loss:
0.1253 - mae: 0.2032 - val_loss: 0.1773 - val_mae: 0.2358
Epoch 43/50
1/1             0s 104ms/step - loss:
0.1124 - mae: 0.1825 - val_loss: 0.1766 - val_mae: 0.2430
Epoch 44/50
1/1             0s 103ms/step - loss:
0.1077 - mae: 0.1702 - val_loss: 0.1764 - val_mae: 0.2457
Epoch 45/50
1/1             0s 108ms/step - loss:
0.1107 - mae: 0.1760 - val_loss: 0.1767 - val_mae: 0.2423
Epoch 46/50
1/1             0s 106ms/step - loss:
0.1069 - mae: 0.1663 - val_loss: 0.1769 - val_mae: 0.2402
Epoch 47/50
1/1             0s 103ms/step - loss:
```

```
0.1124 - mae: 0.1890 - val_loss: 0.1784 - val_mae: 0.2299
Epoch 48/50
1/1             0s 109ms/step - loss:
0.1168 - mae: 0.2042 - val_loss: 0.1813 - val_mae: 0.2172
Epoch 49/50
1/1             0s 103ms/step - loss:
0.1126 - mae: 0.1839 - val_loss: 0.1851 - val_mae: 0.2056
Epoch 50/50
1/1             0s 104ms/step - loss:
0.1233 - mae: 0.2017 - val_loss: 0.1892 - val_mae: 0.1961
Test MSE: 4.0235, Test MAE: 0.6959
```

[3]:
```
['scaler_y.pkl']
```

[4]:
```python
# 7. Prediction and Visualization
def predict_temperature_series(Fb_set, v_set, mu_m, t2):
    """Predict resampled temperature series"""
    #model = keras.models.load_model('~/Documents/pdc_fem_cfd_python_cpp/
 ↪project/brake/nerual/training/brake_temp_model.keras')
    model = keras.models.load_model('brake_temp_model.keras')
    scaler_X = joblib.load('scaler_X.pkl')
    scaler_y = joblib.load('scaler_y.pkl')

    input_data = scaler_X.transform(np.array([[Fb_set, v_set, mu_m, t2]]))
    prediction = model.predict(input_data)
    return scaler_y.inverse_transform(prediction.reshape(1, -1))[0]

def plot_comparison(Fb_set, v_set, mu_m, t2):
    """Compare actual vs predicted with correct time scaling"""
    # Get prediction
    pred = predict_temperature_series(Fb_set, v_set, mu_m, t2)

    # Find matching test data
    actual_series = None
    for test in test_data:
        if (np.isclose(test['Fb_set'], Fb_set) and
            np.isclose(test['v_set'], v_set) and
            np.isclose(test[' _m'], mu_m) and
            np.isclose(test['t2'], t2)):
            actual_series = test['temperature_series']
            break

    # Create time axes
    pred_time = np.linspace(0, t2, TARGET_LENGTH)
    if actual_series is not None:
        actual_time = np.linspace(0, t2, len(actual_series))
```

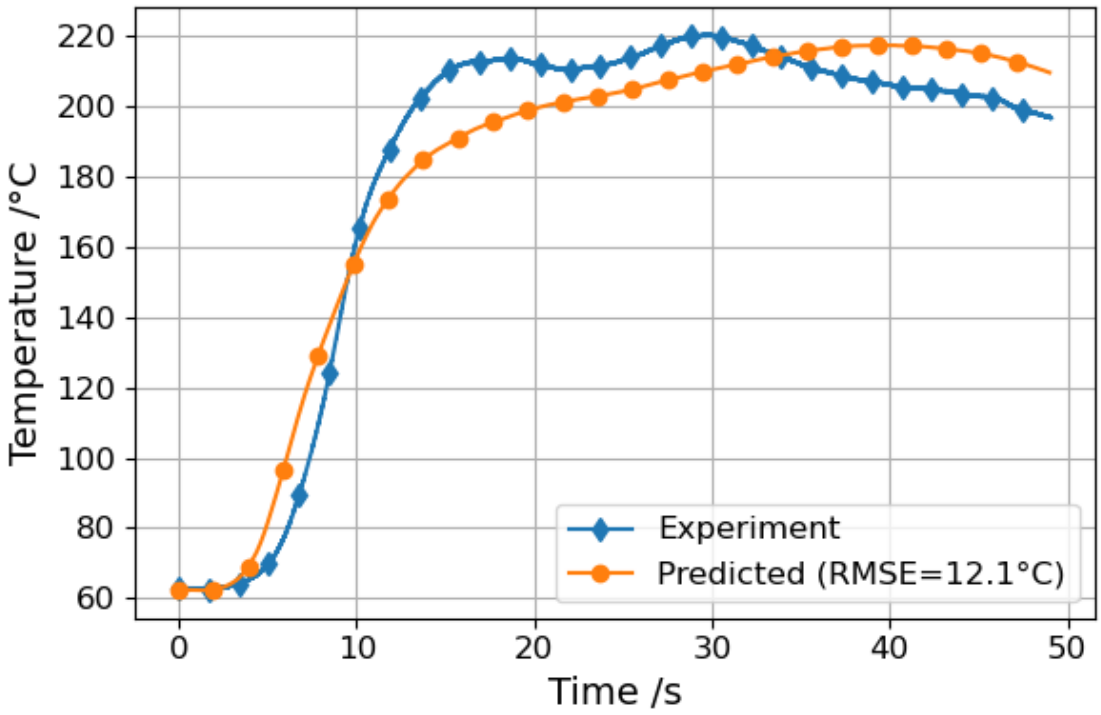
```
    # Plot
    plt.figure(figsize=(10, 5))
    if actual_series is not None:
        plt.plot(actual_time, actual_series, 'b-', label='Actual', linewidth=2)
    plt.plot(pred_time, pred, 'r--', label='Predicted', linewidth=2)

    plt.title(f'Temperature Profile\n(Fb={Fb_set} kN, v={v_set} km/h,  ={mu_m},␣
 ↪t2={t2}s)')
    plt.xlabel('Time (s)')
    plt.ylabel('Temperature (°C)')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# Example usage
#if len(test_data) > 0:
#    sample = test_data[0]
#   ␣
 ↪plot_comparison(sample['Fb_set'],sample['v_set'],sample['_m'],sample['t2'])
```

[8]:
```
import numpy as np
from sklearn.metrics import mean_squared_error

# 1. Get predictions and experimental data
pred = predict_temperature_series(23, 160, 0.376, 49.1)
pd1 = pd.read_csv('4.csv')
T_expe = np.array(pd1['ave'])

# 2. Resample experimental data to match prediction length
def resample_to_target(original_series, original_length, target_length):
    """Resample using linear interpolation"""
    original_time = np.linspace(0, 1, original_length)
    target_time = np.linspace(0, 1, target_length)
    return np.interp(target_time, original_time, original_series)

T_expe_resampled = resample_to_target(
    T_expe,
    len(T_expe),
    len(pred)
)

# 3. Calculate RMSE
rmse = np.sqrt(mean_squared_error(pred, T_expe_resampled))
print(f"RMSE: {rmse:.2f} °C")

# 4. Add to plot
```

```
plt.figure(figsize=(6, 4))
time_expe = np.arange(len(T_expe)) * (49.1/len(T_expe))
plt.plot(time_expe, T_expe, label='Experiment',marker='d',markevery=1000)

time_pred = np.arange(len(pred)) * (49.1/len(pred))
plt.plot(time_pred, pred, label=f'Predicted (RMSE={rmse:.1f}°C)',marker='o',␣
 ↪markevery=20)

glb = 14

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
#plt.title('Fb=23 kN, v=160 km/h,  =0.376, braking time=49 s')
plt.xlabel('Time /s', fontsize=glb)
plt.ylabel('Temperature /°C', fontsize=glb)
plt.legend(fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.savefig('machine_12.1.pdf')
```

```
/Users/yanjun/Documents/apps/miniconda3/envs/dd2421/lib/python3.10/site-
packages/keras/src/saving/saving_lib.py:802: UserWarning: Skipping variable
loading for optimizer 'rmsprop', because it has 9 variables whereas the saved
optimizer has 16 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
```

```
1/1             1s 868ms/step
RMSE: 12.08 °C
```



2025-6-19, until now, the best RMSE is 13.2

[ ]:

[ ]: