

نام استاد: دکتر فصحتی

نام درس: مدارهای واسط

شماره دانشجویی: ۴۰۱۱۰۶۳۶۳

نام دانشجو: میترا قلی پور

شماره دانشجویی: ۴۰۱۱۰۶۲۵۵

نام دانشجو: ملیکا علیزاده

شماره دانشجویی: ۴۰۱۱۷۰۶۶۱

نام دانشجو: الینا هژبری

۱- مقدمه

در بسیاری از پروژه‌های صنعتی و تحقیقاتی، نظارت بر دما از اهمیت بالایی برخوردار است. به‌ویژه در سیستم‌های باتری، دما نقش مهمی در عملکرد و عمر باتری‌ها ایفا می‌کند. اما در بسیاری از موارد، استفاده از سنسور دما به دلایل مختلفی از جمله محدودیت‌های اندازه، هزینه‌های بالا، و مشکلات طراحی به‌صرفه نیست. بنابراین، استفاده از روش‌های پیش‌بینی دما بر اساس داده‌های موجود، یک راه‌حل جایگزین و موثر به‌شمار می‌آید.

در این پروژه، با استفاده از داده‌های ولتاژ، جریان، وضعیت و زمان باتری‌ها، هدف پیش‌بینی دمای آن‌ها است. با طراحی یک مدل یادگیری ماشین مبتنی بر شبکه عصبی، سعی شده است تا از ویژگی‌های مختلف داده‌ها برای پیش‌بینی دما به‌طور دقیق و با استفاده از داده‌های تاریخی استفاده شود. این روش نه تنها هزینه‌های مرتبط با استفاده از سنسورهای فیزیکی را کاهش می‌دهد، بلکه به دقت بالا و قابلیت توسعه در سیستم‌های پیچیده‌تر کمک می‌کند.

۲- شرح پروژه

در این پروژه، هدف این است که یک مدل شبکه عصبی روی بورد Arduino (با استفاده از مدل ESP32 WROOM) پیاده‌سازی شود تا دمای باتری پیش‌بینی شود. داده‌های ورودی به مدل از طریق دو پروتکل مختلف WiFi و سریال منتقل می‌شوند. برای پردازش و پیش‌بینی دما، از یک مدل یادگیری ماشین مبتنی بر شبکه عصبی استفاده شده است که ابتدا با استفاده از TensorFlow طراحی و آموزش داده شده است.

پس از آموزش مدل، آن را به یک کتابخانه C تبدیل کرده‌ایم تا بتوان آن را روی برد ESP32 WROOM بارگذاری کرد. با استفاده از این کتابخانه، داده‌های ورودی به برد ارسال شده و مدل پیش‌بینی دمای باتری را محاسبه کرده و خروجی آن از طریق پروتکل‌های WiFi یا سریال به سیستم‌های دیگر ارسال می‌شود. این روش شبیه سازی از طراحی برد هایی که می‌توانند جایگزین سنسورهای دما در باتری شوند.

۳- آموزش مدل شبکه عصبی

نحوه اجرای یادگیری شبکه عصبی را می‌توانید در آدرس “make_model/train.ipynb” ببینید. در این فایل، مدل پیش‌بینی دما بر اساس ولتاژ و جریان باتری طراحی شده است. ابتدا داده‌های موجود در فایل CSV بارگذاری شده و پیش‌پردازش‌هایی روی آن‌ها انجام شد. این پیش‌پردازش‌ها شامل حذف ستون‌های غیر ضروری و تبدیل ویژگی state به فرمت one-hot encoding بود. سپس، داده‌ها به ویژگی‌ها (X) و هدف (y) تقسیم شده و داده‌های ویژگی مقیاس‌بندی نشده برای آموزش و ارزیابی مدل آماده شد.

مدل شبکه عصبی با استفاده از TensorFlow ساخته شده است. مدل دارای دو لایه مخفی با تعداد نوروهای ۳۲ و ۱۶ و یک لایه خروجی برای پیش‌بینی دما است. مدل با استفاده از داده‌های آموزشی آموزش داده شده و سپس بر روی داده‌های آزمایشی ارزیابی شد.

بعد از آموزش مدل، پیش‌بینی‌هایی بر اساس داده‌های آزمایشی انجام شد و نمودار مقایسه‌ای بین مقادیر پیش‌بینی شده و مقادیر واقعی رسم شد. سپس، مدل به فرمت TensorFlow Lite تبدیل شد تا بتوان از آن در برد آردوینو استفاده کرد.

در نهایت، مدل TensorFlow Lite بر روی داده‌های تست اجرا شده و پیش‌بینی دما برای هر نمونه از داده‌ها نمایش داده شد. برای دیتاست نیز از دیتاست ناسا استفاده شده است.

۴- پیاده‌سازی پروژه با استفاده از پروتکل سریال

برای انجام این پروژه، ما از پروتکل سریال برای انتقال داده‌ها بین کامپیوتر و برد ESP32 استفاده کردیم. در ابتدا، مدل شبکه عصبی که پیش‌بینی دمای باتری را انجام می‌دهد، به برد منتقل شد و سپس داده‌ها از کامپیوتر به برد ارسال می‌شود تا مدل بتواند آن‌ها را پردازش کرده و دمای پیش‌بینی شده را برگرداند.

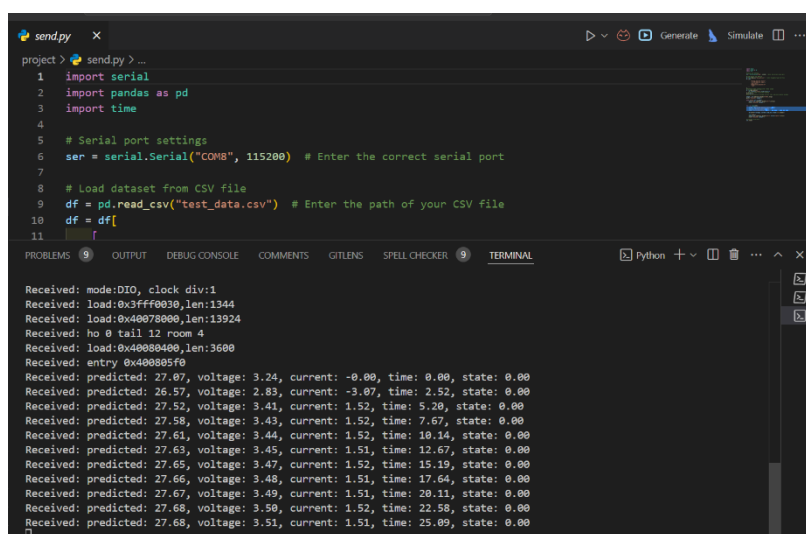
مراحل انجام پروژه:

- ۱- بارگذاری مدل شبکه عصبی: مدل شبکه عصبی که برای پیش‌بینی دما طراحی شده است، به صورت یک فایل آرایه‌ای تبدیل شده و در برد ESP32 بارگذاری می‌شود. این کار با استفاده از توابع کتابخانه TensorFlow Lite_ESP32 انجام می‌شود.
- ۲- مقداردهی اولیه: ابتدا متغیرهای لازم برای ذخیره داده‌ها (ولتاژ، جریان، زمان و وضعیت باتری) تعریف شده است. این داده‌ها به عنوان ورودی به مدل شبکه عصبی ارسال می‌شوند. در این مرحله، حافظه برای مدل تخصیص داده می‌شود و از توابع AllocateTensors() و Interpreter برای آماده‌سازی محیط اجرا استفاده می‌شود.

۳- دریافت داده‌ها از پورت سریال: در حلقه اصلی، هر بار که داده‌ای از کامپیوتر از طریق پروتکل سریال دریافت می‌شود، این داده‌ها به فرمت مناسب پردازش می‌شوند. داده‌ها شامل ولتاژ، جریان، زمان و وضعیت باتری هستند که در قالب یک رشته متنی از هم جدا شده‌اند. سپس این داده‌ها استخراج شده و به متغیرهای مختلف اختصاص داده می‌شوند.

۴- پردازش داده‌ها و پیش‌بینی دما: داده‌ها پس از استخراج، در آرایه‌ای به نام `input_data` قرار می‌گیرند. این آرایه به ورودی مدل ارسال می‌شود. سپس مدل با استفاده از تابع `Invoke()` اجرا می‌شود تا پیش‌بینی دمای باتری انجام شود. نتیجه پیش‌بینی شده در متغیر `predictedTemperature` ذخیره می‌شود.

۵- ارسال خروجی به کامپیوتر: پس از پردازش داده‌ها، دمای پیش‌بینی شده به همراه مقادیر ورودی (ولتاژ، جریان، زمان و وضعیت) به صورت یک رشته متنی به کامپیوتر ارسال می‌شود. این داده‌ها از طریق پورت سریال ارسال و در کامپیوتر نمایش داده می‌شوند.



```
1 import serial
2 import pandas as pd
3 import time
4
5 # Serial port settings
6 ser = serial.Serial("COM8", 115200) # Enter the correct serial port
7
8 # Load dataset from CSV file
9 df = pd.read_csv("test_data.csv") # Enter the path of your CSV file
10 df = df[
11     ]
12
13 Received: mode:DIO, clock div:1
14 Received: load:0x3fff0030,len:1344
15 Received: load:0x40078000,len:13924
16 Received: ho @ tail 12 room 4
17 Received: load:0x40080400,len:3600
18 Received: entry 0x400805f0
19 Received: predicted: 27.07, voltage: 3.24, current: -0.00, time: 0.00, state: 0.00
20 Received: predicted: 26.57, voltage: 2.83, current: -3.07, time: 2.52, state: 0.00
21 Received: predicted: 27.52, voltage: 3.41, current: 1.52, time: 5.20, state: 0.00
22 Received: predicted: 27.56, voltage: 3.43, current: 1.52, time: 7.67, state: 0.00
23 Received: predicted: 27.61, voltage: 3.44, current: 1.52, time: 10.14, state: 0.00
24 Received: predicted: 27.63, voltage: 3.45, current: 1.51, time: 12.67, state: 0.00
25 Received: predicted: 27.65, voltage: 3.47, current: 1.52, time: 15.19, state: 0.00
26 Received: predicted: 27.66, voltage: 3.48, current: 1.51, time: 17.64, state: 0.00
27 Received: predicted: 27.67, voltage: 3.49, current: 1.51, time: 20.11, state: 0.00
28 Received: predicted: 27.68, voltage: 3.50, current: 1.52, time: 22.58, state: 0.00
29 Received: predicted: 27.68, voltage: 3.51, current: 1.51, time: 25.09, state: 0.00
```

شکل ۱- خروجی حاصل از انتقال با پروتکل `serial` سمت ارسال کننده داده

۵- پیاده‌سازی پروژه با استفاده از پروتکل WiFi

برای پیاده‌سازی این پروژه با استفاده از پروتکل WiFi، ابتدا دستگاه ESP32 به شبکه WiFi متصل می‌شود. اطلاعات مربوط به SSID و رمز عبور شبکه در کد تنظیم شده و سپس از تابع `WiFi.begin(ssid, password)` برای شروع اتصال استفاده می‌شود. پس از اتصال به شبکه، آدرس IP دستگاه بر روی سریال نمایش داده می‌شود. بعد از این مرحله، یک سرور HTTP در پورت ۸۰ راه‌اندازی می‌شود که برای دریافت درخواست‌های ورودی از کلاینت‌ها آماده است.

در قسمت بعدی، زمانی که یک درخواست HTTP از کلاینت دریافت می‌شود، پارامترهای مربوط به ولتاژ، جریان، زمان و وضعیت به سرور ارسال می‌شود. این پارامترها به مدل ورودی ارسال شده و پیش‌بینی دما از طریق مدل انجام می‌شود. پس از اجرای مدل، دما پیش‌بینی شده به صورت یک پاسخ HTTP به کلاینت ارسال می‌شود. این فرایند برای هر درخواست به صورت تکراری انجام می‌شود و تأخیر ۱ ثانیه بین درخواست‌ها اعمال می‌شود تا سرور دچار تداخل نشود.

```
Output Serial Monitor X
Message (Enter to send message to "ESP32-WROOM-DA Module" on "COM8")
New Line 115200 baud

Received request: GET /predict?voltage=3.63904317087447&current=1.51544162877764&time=72.563&state=0.0 HTTP/1.1
Received request: GET /predict?voltage=3.64425295575452&current=1.51448477286817&time=75.016&state=0.0 HTTP/1.1
Received request: GET /predict?voltage=3.64937342376991&current=1.5132726369347&time=77.485&state=0.0 HTTP/1.1
Received request: GET /predict?voltage=3.65462748264284&current=1.51266126565704&time=79.954&state=0.0 HTTP/1.1
Received request: GET /predict?voltage=3.6597100742432&current=1.51485266486728&time=82.469&state=0.0 HTTP/1.1
Received request: GET /predict?voltage=3.66311408112&current=1.5156155543666&time=167.422&state=0.0 HTTP/1.1
```

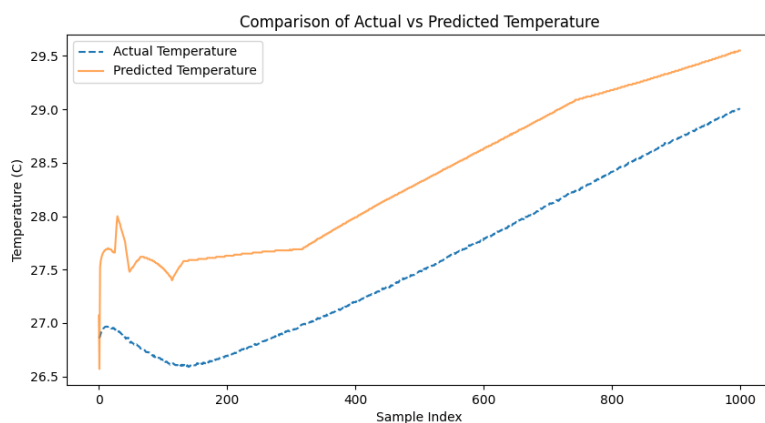
شکل ۲- خروجی حاصل از انتقال داده با پروتکل WiFi سمت ESP32

```
PROBLEMS 55 OUTPUT DEBUG CONSOLE COMMENTS GITLENS SPELL CHECKER 46 TERMINAL python + v [ ] ... ^ x

Response: 27.94
Sent data: {'voltage': 3.63904317087447, 'current': 1.51544162877764, 'time': 72.563, 'state': 0.0}
Response: 28.00
Sent data: {'voltage': 3.64425295575452, 'current': 1.51448477286817, 'time': 75.016, 'state': 0.0}
Response: 27.98
Sent data: {'voltage': 3.64937342376991, 'current': 1.5132726369347, 'time': 77.485, 'state': 0.0}
Response: 27.96
Sent data: {'voltage': 3.65462748264284, 'current': 1.51266126565704, 'time': 79.954, 'state': 0.0}
Response: 27.94
Sent data: {'voltage': 3.6597100742432, 'current': 1.51485266486728, 'time': 82.469, 'state': 0.0}
Response: 27.93
```

شکل ۳- خروجی حاصل از انتقال داده با پروتکل WiFi سمت ارسال کننده داده

۶- تحلیل داده‌ها



شکل ۴- مقایسه دمای واقعی و دمای پیش‌بینی شده

```
Received: 29.50
Received: 29.50
Received: 29.51
Received: 29.51
Received: 29.51
Received: 29.51
Received: 29.52
Received: 29.52
Received: 29.52
Received: 29.52
Received: 29.53
Received: 29.53
Received: 29.53
Received: 29.53
Received: 29.53
Received: 29.54
Received: 29.53
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.55
Received: 29.55
Received: 29.55
Total transfer time: 7.12 seconds
Mean Squared Error (MSE): 0.64463
```

شکل ۵- محاسبه زمان ارسال ۱۰۰۰ داده و خطای میانگین مربعات آن‌ها برای پروتکل serial

```
Received: 29.53
Received: 29.53
Received: 29.53
Received: 29.53
Received: 29.53
Received: 29.54
Received: 29.53
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.54
Received: 29.55
Received: 29.55
Received: 29.55
Total transfer time: 30.29 seconds
Mean Squared Error (MSE): 0.64425
```

شکل ۶- محاسبه زمان ارسال ۱۰۰۰ داده و خطای میانگین مربعات آن‌ها برای پروتکل WiFi

۷- مدل‌های شکست‌خورده

۱. مدل اول: این مدل، یک مدل شبکه عصبی کانولوشنی (CNN) است که با استفاده از کتابخانه‌های TensorFlow و Keras، دمای باتری را از طریق داده‌های مربوط به ولتاژ، جریان و زمان پیش‌بینی می‌کند. در این مدل، ابتدا داده‌های موجود در فایل CSV بارگذاری می‌شود و سپس با استفاده از pad_sequences از کتابخانه Keras، داده‌ها به هم تراز شده تا برای مدل ورودی مناسب شوند.

ورودی‌ها را به دو بخش X و Y تفکیک می‌کنیم؛ X شامل ویژگی‌هایی مانند ولتاژ، جریان، زمان و وضعیت باتری است که به مدل داده می‌شود و Y شامل دمای باتری است که مدل باید پیش‌بینی کند.

برای ساخت مدل CNN، یک مدل Sequential ایجاد می‌شود که ابتدا یک لایه masking اضافه می‌شود تا ورودی‌هایی با مقدار صفر نادیده گرفته شوند. سپس دو لایه کانولوشنی Conv1D اضافه می‌شود که از نورون‌های ۱۶ و ۳۲ استفاده می‌کند و از relu به عنوان تابع فعال‌سازی بهره می‌برند.

'padding = 'casual' به مدل اجازه می‌دهد که پیش‌بینی‌هایی با توجه به داده‌های قبلی داشته باشد. سپس لایه‌های LayerNormalization و Dropout برای نرمال‌سازی داده‌ها و جلوگیری از overfitting اضافه می‌شوند. در آخر لایه dense با یک نود به مدل اضافه می‌شود تا پیش‌بینی دما را نشان دهد.

این مدل با بهینه‌ساز Adam و معیار خطای میانگین مربعات (MSE) کامپایل می‌شود و پیش‌بینی دما را چاپ می‌کند.

```
31 # Build a CNN model considering temporal dependencies
32 model = Sequential()
33 model.add(Masking(mask_value=0.0, input_shape=(X_train.shape[1], X_train.shape[2])))
34
35 # 1D Convolutional layers with Causal Padding
36 model.add(Conv1D(filters=16, kernel_size=3, activation='relu', padding='causal'))
37 model.add(Conv1D(filters=32, kernel_size=3, activation='relu', padding='causal'))
38
39 # Layer normalization and dropout for improved performance
40 model.add(LayerNormalization())
41 model.add(Dropout(0.2))
42
43 # Output layer with time-sequential structure
44 model.add(Dense(1)) # Output preserves the temporal structure of the input
45
46 # Compile the model
47 model.compile(optimizer=Adam(learning_rate=0.001), loss="mse")
48 model.summary()
49
50 # Train the model
51 history = model.fit(
52     X_train, y_train, epochs=300, batch_size=32, validation_data=(X_val, y_val)
53 )
54
55 # Evaluate the model
56 loss = model.evaluate(X_val, y_val)
57 print(f"Validation Loss: {loss}")
58
```

```

31 # Build a CNN model considering temporal dependencies
32 model = Sequential()
33 model.add(Masking(mask_value=0.0, input_shape=(X_train.shape[1], X_train.shape[2])))
34
35 # 1D Convolutional layers with Causal Padding
36 model.add(Conv1D(filters=16, kernel_size=3, activation='relu', padding='causal'))
37 model.add(Conv1D(filters=32, kernel_size=3, activation='relu', padding='causal'))
38
39 # Layer normalization and dropout for improved performance
40 model.add(LayerNormalization())
41 model.add(Dropout(0.2))
42
43 # Output layer with time-sequential structure
44 model.add(Dense(1)) # Output preserves the temporal structure of the input
45
46 # Compile the model
47 model.compile(optimizer=Adam(learning_rate=0.001), loss="mse")
48 model.summary()
49
50 # Train the model
51 history = model.fit(
52     X_train, y_train, epochs=300, batch_size=32, validation_data=(X_val, y_val)
53 )
54
55 # Evaluate the model
56 loss = model.evaluate(X_val, y_val)
57 print(f"Validation Loss: {loss}")
58
59 # Predict for a sequence of data
60 predicted_temperature = model.predict(X_val[:1])
61 print(f"Predicted Temperature: {predicted_temperature.shape}")
62 print(f"True Temperature: {y_val[0, -3:, 0]}")
63
64 # Save the model
65 model.save("cnn_temporal_model.h5")

```

شکل ۷- کد مدل شکست خورده ۱

۲. مدل دوم: مدل LSTM، یک مدل شبکه عصبی بازگشتی است که برای داده‌های دنباله‌ای استفاده می‌شود که در آن مقادیر قبلی تأثیر زیادی بر پیش‌بینی مقادیر بعدی دارند.

در این مدل، ابتدا داده‌های موجود در فایل CSV بارگذاری می‌شود و سپس داده‌های جمع‌آوری شده با استفاده از `pad_sequences`، `padding` می‌شود تا طول تمام دنباله‌ها یکسان شود. این عمل باعث می‌شود که دنباله‌های کوتاه‌تر با صفر در انتها پر شود.

ورودی‌ها را به دو بخش X و Y تفکیک می‌کنیم؛ X شامل ویژگی‌هایی مانند ولتاژ، جریان، زمان و وضعیت باتری است که به مدل داده می‌شود و Y شامل دمای باتری است که مدل باید پیش‌بینی کند.

ابتدا لایه `masking` برای پردازش دنباله‌هایی با طول متغیر و نادیده گرفتن صفرهای `padding` اضافه می‌شود. سپس لایه LSTM را داریم که برای پردازش دنباله‌ها استفاده می‌شود و برای فعال‌سازی آن از `relu` استفاده می‌شود. همچنین با `return_sequences=True` تضمین می‌شود که مدل برای هر گام زمانی خروجی را برمی‌گرداند. در آخر لایه `dense` اضافه می‌شود که یک لایه تمام متصل با یک نورون برای پیش‌بینی دما است.

مدل با استفاده از بهینه‌ساز Adam با نرخ یادگیری ۰/۰۰۱ و خطای میانگین مربعات (MSE)، به عنوان تابع هزینه کامپایل می‌شود.

```

1 import numpy as np
2 import tensorflow as tf
3 import pandas as pd
4 import os
5 from tensorflow.keras.models import Sequential
6 from tensorflow.keras.layers import LSTM, Dense, Masking
7 from tensorflow.keras.optimizers import Adam
8 from sklearn.model_selection import train_test_split
9 from tensorflow.keras.preprocessing.sequence import pad_sequences
10 from sklearn.preprocessing import StandardScaler
11 import matplotlib.pyplot as plt
12
13 data = []
14 count_battery = 6
15 for i in range(count_battery):
16     for folder in sorted(os.listdir("./dataset/" + str(i + 1) + "/" + "/")):
17         if not folder.startswith("B0"):
18             continue
19
20         root = os.path.join('dataset', str(i + 1), folder, 'csv')
21         for file in sorted(os.listdir(root)):
22             if '_charge' in file:
23                 a = pd.read_csv(os.path.join(root, file))
24                 if a.isnull().values.any():
25                     continue
26                 a = a.to_numpy()
27                 data.append(a[10:1000])
28
29 data = pad_sequences(data, padding='post', dtype='float32')
30
31 print(data.shape)
32
33 # Split the data into input (voltage, current) and target (temperature)
34 X = data[:, :, (0,1,3,4)] # voltage and current
35 y = data[:, :, 2:3] # temperature
36 # Split into training and validation sets
37 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
38
39 print(X_train[0,:5,:])
40
41 # Define the LSTM model
42 model = Sequential()
43
44 # Masking layer to handle varying lengths of time series (if any)
45 model.add(Masking(mask_value=0.0, input_shape=(X_train.shape[1], X_train.shape[2])))
46
47 # LSTM layer
48 model.add(LSTM(5, activation="relu", return_sequences=True))
49
50 # Dense layer to predict temperature
51 model.add(Dense(1))
52
53 # Compile the model
54 model.compile(optimizer=Adam(learning_rate=0.001), loss="mse")
55
56 # Print the model summary
57 model.summary()
58
59 print(X_train.shape , y_train.shape)
60
61 # Train the model
62 history = model.fit(
63     X_train, y_train, epochs=300, batch_size=32, validation_data=(X_val, y_val)
64 )
65
66 # Evaluate the model
67 loss = model.evaluate(X_val, y_val)
68 print(f"Validation Loss: {loss}")
69 # New voltage and current data
70 predicted_temperature = model.predict(X_val[:1])
71 print(predicted_temperature.shape)
72 print(f"Predicted Temperature: {predicted_temperature[0,-3:,0]}")
73 print(f"True Temperature: {y_val[0,-3:,0]}")
74
75 model.save_weights("./model_weights_charge.tf")
76 # model.load_weights("./model_weights_charge.tf")

```

شکل ۸- کد مدل شکست خورده ۲

دلیل شکست این دو مدل

به صورت کلی ما مدل‌ها را به مدل TensorFlow_Lite تبدیل می‌کنیم و سپس برای آن‌که روی بورد قابل استفاده باشد، آن را به یک آرایه C تبدیل می‌کنیم. دو مدل شکست خورده بعد از انتقال آرایه به بورد نمی‌توانستند اجرا شوند زیرا در فرآیند تبدیل به مدل opcode از `tf.lite` استفاده می‌شود که کتابخانه `TensorFlow_Lite` ESP32 از آن‌ها پشتیبانی نمی‌کند.

۸- فایل‌های پروژه

کدهای مربوط به یادگیری شبکه عصبی در پوشه `make_model`

کدهای مربوط به انتقال داده با پروتکل سریال در پوشه `Serial`:

- کد ارسال کننده: `send.py`
- کد آردوینو در پوشه `main`

کدهای مربوط به انتقال داده با پروتکل WiFi در پوشه `WiFi`:

- همانند بخش سریال

کدهای مربوط به مدل‌های شکست‌خورده در `failed_models`:

- کد مدل ۱: `failed model 1`
- کد مدل ۲: `failed model 2`