HOMEWORK-4 REPORT

CSE 421

Efe Kayadelen
Student ID : 20220701088

Samet Alper Özdemir
Student ID : 20210702021

GITHUB REPO: https://github.com/blackwh1p/CSE421-Homeworks

Part-1 (Q1):

main.py Code:

This code script implements a Human Activity Recognition pipeline using a Multi-Layer Perceptron (MLP) neural network. It begins by suppressing system warnings and establishing absolute file paths to load the WISDM dataset, which it then splits into training and testing sets based on specific user IDs. The code utilizes custom utility functions to extract features and prepares the data via One-Hot Encoding before training a three-layer dense neural network to classify six different physical activities. Finally, it evaluates the model's performance using a confusion matrix visualization and saves the trained model as an .h5 file for future use.

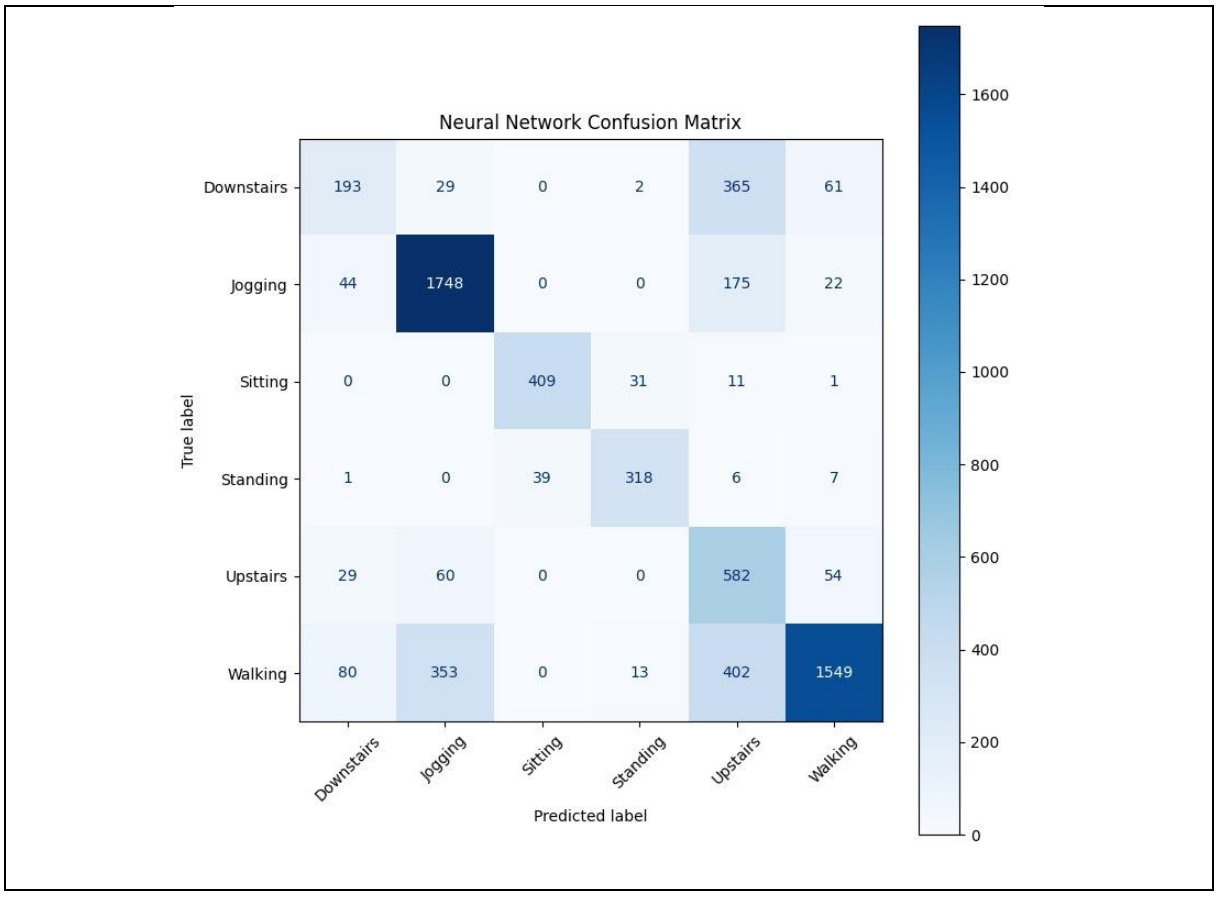
Code:

```
import os
import sys

# --- 1. SUPPRESS WARNINGS (Must be done before importing TensorFlow) ---
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # Suppress TF info/warnings

import warnings
warnings.filterwarnings("ignore") # Suppress Python warnings (like np.object)

# --- IMPORTS ---
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import OneHotEncoder

# Import local utility functions
from data_utils import read_data
from feature_utils import create_features

# --- CONFIGURATION & PATH FIX ---
# Get the directory where THIS script (main.py) is located
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))

# Construct the absolute path to the data file
# This works regardless of where you run the terminal command from
WISDM_PATH = os.path.join(SCRIPT_DIR, "data", "WISDM_ar_v1.1_raw.txt")
MODEL_DIR = os.path.join(SCRIPT_DIR, "Models")

# Ensure model directory exists
if not os.path.exists(MODEL_DIR):
    os.makedirs(MODEL_DIR)

TIME_PERIODS = 80
STEP_DISTANCE = 40

def main():
    print(f"Looking for data at: {WISDM_PATH}")

    # 1. Read Data
```

```
data_df = read_data(WISDM_PATH)

if data_df is None:
    print("¥n!!! FILE ERROR !!!")
    print(f"Python could not find the file at: {WISDM_PATH}")
    print("Please check:")
    print("1. Is the folder named 'data' or 'Data'? (Match the case)")
    print("2. Is the file named 'WISDM_ar_v1.1_raw.txt'?")
    print("3. Does the file have a hidden double extension (e.g. .txt.txt)?")
    return

# 2. Split Data (Train on User IDs <= 28, Test on > 28)
df_train = data_df[data_df["user"] <= 28]
df_test = data_df[data_df["user"] > 28]

print(f"Training samples: {len(df_train)}")
print(f"Testing samples: {len(df_test)}")

# 3. Create Features
print("Extracting features from training data...")
train_segments_df, train_labels = create_features(df_train, TIME_PERIODS, STEP_DISTANCE)

print("Extracting features from testing data...")
test_segments_df, test_labels = create_features(df_test, TIME_PERIODS, STEP_DISTANCE)

# 4. Prepare Data for Neural Network
train_segments_np = train_segments_df.to_numpy()
test_segments_np = test_segments_df.to_numpy()

# One Hot Encoding for Labels
ohe = OneHotEncoder(sparse_output=False)
train_labels_ohe = ohe.fit_transform(train_labels.reshape(-1, 1))

# Encode test labels for evaluation (get integer indices)
categories, test_labels_idx = np.unique(test_labels, return_inverse=True)

print(f"Input features shape: {train_segments_np.shape}")
print(f"Classes: {categories}")

# 5. Define Model
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(10,)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(6, activation="softmax") # 6 Activities
])

model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(1e-3),
              metrics=['accuracy'])

# 6. Train
print("Starting training...")
history = model.fit(train_segments_np, train_labels_ohe,
                    epochs=50,
                    verbose=1,
                    validation_split=0.1)
```

```
    # 7. Evaluate
    print("Evaluating on test set...")
    nn_preds = model.predict(test_segments_np)
    predicted_classes = np.argmax(nn_preds, axis=1)

    # 8. Confusion Matrix
    conf_matrix = confusion_matrix(test_labels_idx, predicted_classes)

    # Plot
    fig, ax = plt.subplots(figsize=(8, 8))
    cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=categories)
    cm_display.plot(ax=ax, cmap='Blues', xticks_rotation=45)
    ax.set_title("Neural Network Confusion Matrix")
    plt.tight_layout()
    plt.show()

    # 9. Save Model
    save_path = os.path.join(MODEL_DIR, "har_mlp.h5")
    model.save(save_path)
    print(f"Model saved to {save_path}")

if __name__ == "__main__":
    main()
```

Result:



Part-2 (Q2):

main.py Code:

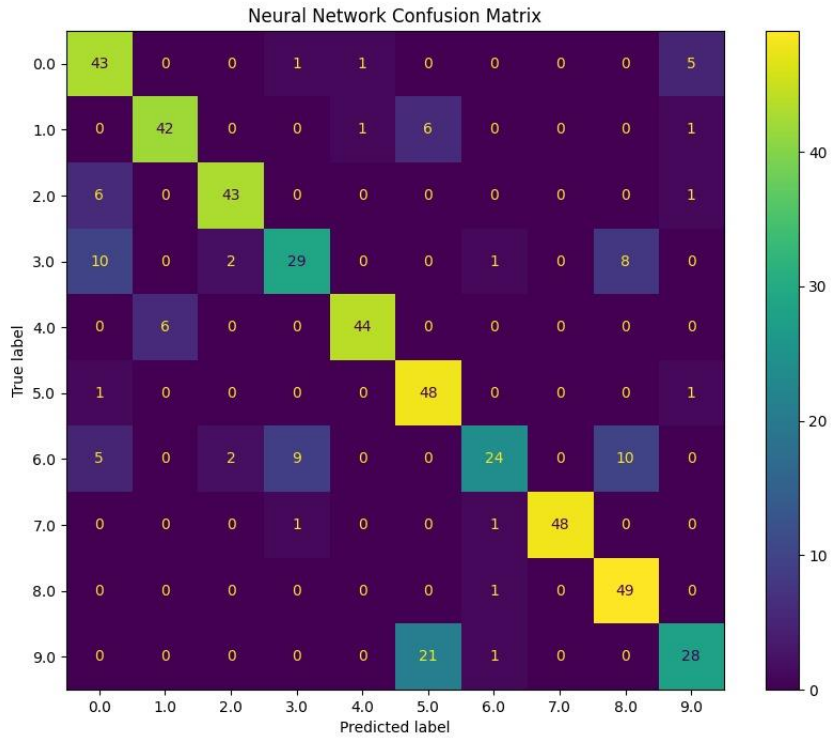
This code script performs Keyword Spotting (KWS) by training a neural network to recognize spoken commands from audio files. It extracts Mel-Frequency Cepstral Coefficients (MFCC) features using a Hamming window and Mel-filter bank, splits the dataset into training and testing sets based on the speaker's name, and feeds the resulting feature vectors into a multi-layer dense neural network. After training for 100 epochs, the script evaluates the model's accuracy using a confusion matrix to visualize classification performance across 10 categories and saves the final model as an .h5 file.

Code:

```
import os
import sys

# --- 1. SUPPRESS WARNINGS (Must be done before importing TensorFlow) ---
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # Suppress TF info/warnings

import warnings
warnings.filterwarnings("ignore") # Suppress Python warnings (like np.object)

import numpy as np
import scipy.signal as sig
import tensorflow as tf
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import OneHotEncoder
from matplotlib import pyplot as plt

# Import local module
from mfcc_func import create_mfcc_features

# --- CONFIGURATION ---
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))

# Data is expected in: project/data/recordings/*.wav
DATA_DIR = os.path.join(SCRIPT_DIR, "data", "recordings")
MODEL_DIR = os.path.join(SCRIPT_DIR, "models")

# Ensure models directory exists
if not os.path.exists(MODEL_DIR):
    os.makedirs(MODEL_DIR)

def main():
    if not os.path.exists(DATA_DIR):
        print(f"Error: Data directory not found at {DATA_DIR}")
        print("Please create 'data/recordings' and place .wav files there.")
        return

    # 1. Load File List
    recordings_list = [os.path.join(DATA_DIR, f) for f in os.listdir(DATA_DIR) if f.endswith('.wav')]
    if not recordings_list:
        print("No .wav files found.")
        return

    # 2. Parameters
    FFTSize = 1024
    sample_rate = 8000
    numOfMelFilters = 20
    numOfDctOutputs = 13

    # CMSIS-DSP expects window function as array
    window = sig.get_window("hamming", FFTSize)
```

```
    # 3. Train/Test Split (Based on speaker 'yweweler' as per book)
    # We filter strings to separate lists
    test_files = [rec for rec in recordings_list if "yweweler" in os.path.basename(rec)]
    train_files = [rec for rec in recordings_list if rec not in test_files]

    print(f"Training samples: {len(train_files)}")
    print(f"Testing samples: {len(test_files)}")

    # 4. Extract Features
    print("Creating Training Features...")
    train_mfcc_features, train_labels = create_mfcc_features(
        train_files, FFTSize, sample_rate, numOfMelFilters, numOfDctOutputs, window
    )

    print("Creating Testing Features...")
    test_mfcc_features, test_labels = create_mfcc_features(
        test_files, FFTSize, sample_rate, numOfMelFilters, numOfDctOutputs, window
    )

    # 5. Define Model
    # Input shape is 26 (13 DCT outputs * 2 frames)
    model = tf.keras.models.Sequential([
        tf.keras.layers.Input(shape=(26,)),
        tf.keras.layers.Dense(100, activation="relu"),
        tf.keras.layers.Dense(100, activation="relu"),
        tf.keras.layers.Dense(10, activation="softmax")
    ])

    # 6. Prepare Data
    ohe = OneHotEncoder(sparse_output=False)
    train_labels_ohe = ohe.fit_transform(train_labels.reshape(-1, 1))

    # Get categories for confusion matrix
    categories, test_labels_idx = np.unique(test_labels, return_inverse=True)

    # 7. Compile and Train
    model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
                  optimizer=tf.keras.optimizers.Adam(1e-3),
                  metrics=['accuracy'])

    model.fit(train_mfcc_features, train_labels_ohe, epochs=100, verbose=1)

    # 8. Evaluate
    nn_preds = model.predict(test_mfcc_features)
    predicted_classes = np.argmax(nn_preds, axis=1)

    # 9. Confusion Matrix
    conf_matrix = confusion_matrix(test_labels, predicted_classes)

    fig, ax = plt.subplots(figsize=(10, 8))
    cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=categories)
    cm_display.plot(ax=ax)
    ax.set_title("Neural Network Confusion Matrix")
    plt.show()

    # 10. Save Model
    save_path = os.path.join(MODEL_DIR, "kws_mlp.h5")
    model.save(save_path)
    print(f"Model saved to {save_path}")

if __name__ == "__main__":
    main()
```

Result:



Part-3 (Q3):

application_11.8.py Code:

This code script implements a Handwritten Digit Recognition system by training an MLP classifier on the MNIST dataset using Hu Moments as shape-based features. Instead of feeding raw pixels into the neural network, it uses OpenCV to calculate seven invariant moments for each digit image, providing a compressed representation of the character's geometry. The model is trained with early stopping and model checkpointing to prevent overfitting and ensure the best version is saved. Finally, it evaluates the system by predicting the test set digits and displaying the results in a confusion matrix.

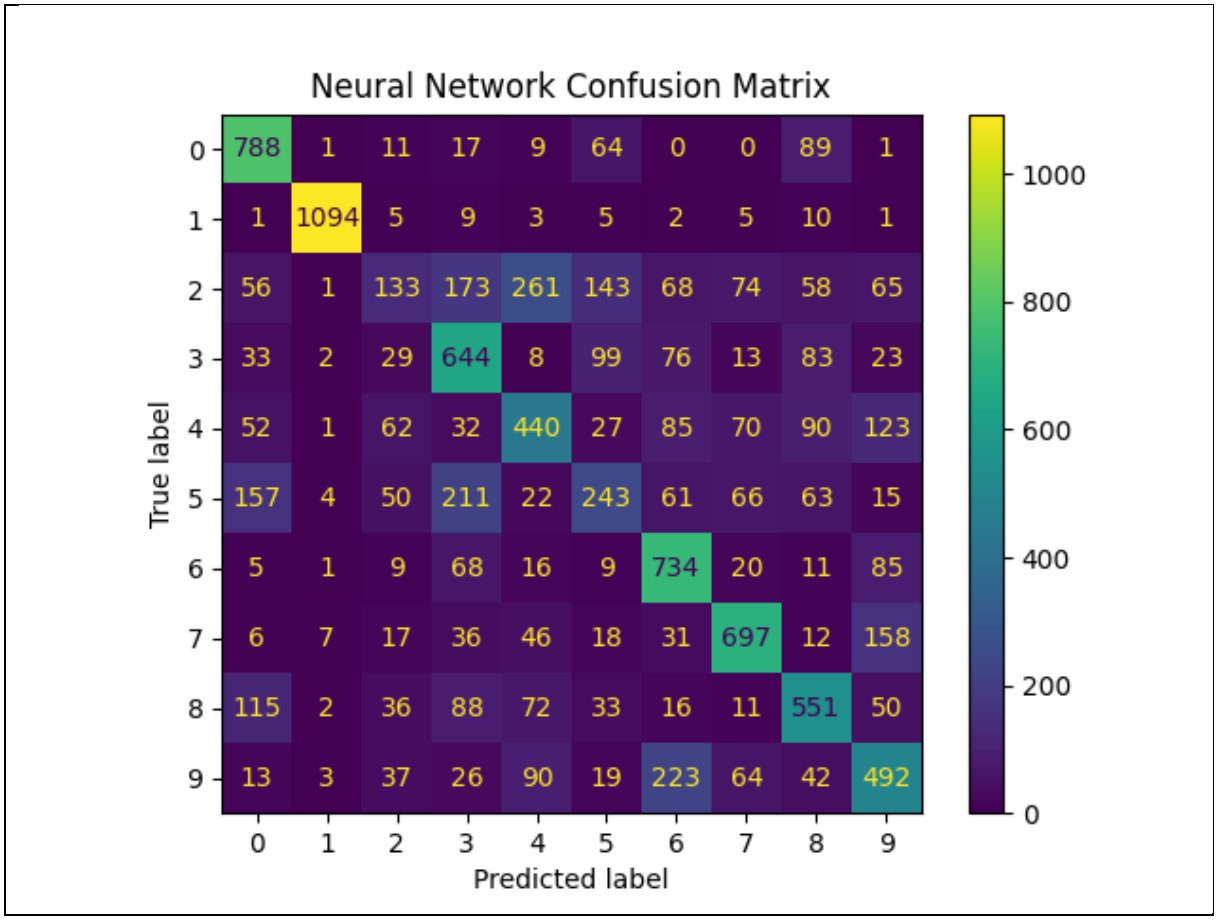
Code:

```
import os
import sys
import numpy as np
import cv2
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import keras
from matplotlib import pyplot as plt

# 1. Setup Paths manually to avoid 'ModuleNotFoundError'
# This defines the 'Models' folder relative to where this script is saved
base_dir = os.path.dirname(os.path.abspath(__file__))
KERAS_MODEL_DIR = os.path.join(base_dir, "Models")
```

```
# Create the folder if it doesn't exist yet
if not os.path.exists(KERAS_MODEL_DIR):
    os.makedirs(KERAS_MODEL_DIR)

# Using .keras extension for better compatibility with Keras 3
model_save_path = os.path.join(KERAS_MODEL_DIR, "hdr_mlp.keras")

# 2. Load Data
(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()

# 3. Feature Extraction (Hu Moments)
train_huMoments = np.empty((len(train_images), 7))
test_huMoments = np.empty((len(test_images), 7))

print("Extracting Hu Moments from training data...")
for train_idx, train_img in enumerate(train_images):
    train_moments = cv2.moments(train_img, True)
    train_huMoments[train_idx] = cv2.HuMoments(train_moments).reshape(7)

print("Extracting Hu Moments from test data...")
for test_idx, test_img in enumerate(test_images):
    test_moments = cv2.moments(test_img, True)
    test_huMoments[test_idx] = cv2.HuMoments(test_moments).reshape(7)

# 4. Define Model
model = keras.models.Sequential([
    keras.layers.Input(shape=(7,)),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])

# 5. Compile and Setup Callbacks
# Accessing callbacks via 'keras.callbacks' avoids the circular import error
model.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(),
    optimizer=keras.optimizers.Adam(learning_rate=1e-4)
)

mc_callback = keras.callbacks.ModelCheckpoint(model_save_path, save_best_only=True, monitor="loss")
es_callback = keras.callbacks.EarlyStopping(monitor="loss", patience=5)

# 6. Training
print("Starting training...")
model.fit(
    train_huMoments,
    train_labels,
    epochs=1000,
    verbose=1,
    callbacks=[mc_callback, es_callback]
)

# 7. Evaluation
# Load the best version saved by the callback
model = keras.models.load_model(model_save_path)
nn_preds = model.predict(test_huMoments)
predicted_classes = np.argmax(nn_preds, axis=1)
```

```
# 8. Visualization
categories = np.unique(test_labels)
conf_matrix = confusion_matrix(test_labels, predicted_classes)
cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=categories)

cm_display.plot()
cm_display.ax_.set_title("Neural Network Confusion Matrix")
plt.show()
```

Result:



Part-4 (Q4):

application_11.9.py Code:

This code script performs time-series temperature forecasting by training a regression model to estimate future room temperatures based on historical data. It downsamples raw sensor data to hourly intervals and uses a sliding window approach to create features from the five previous temperature readings (t-5 to t-1). After normalizing the data, it trains a deep neural

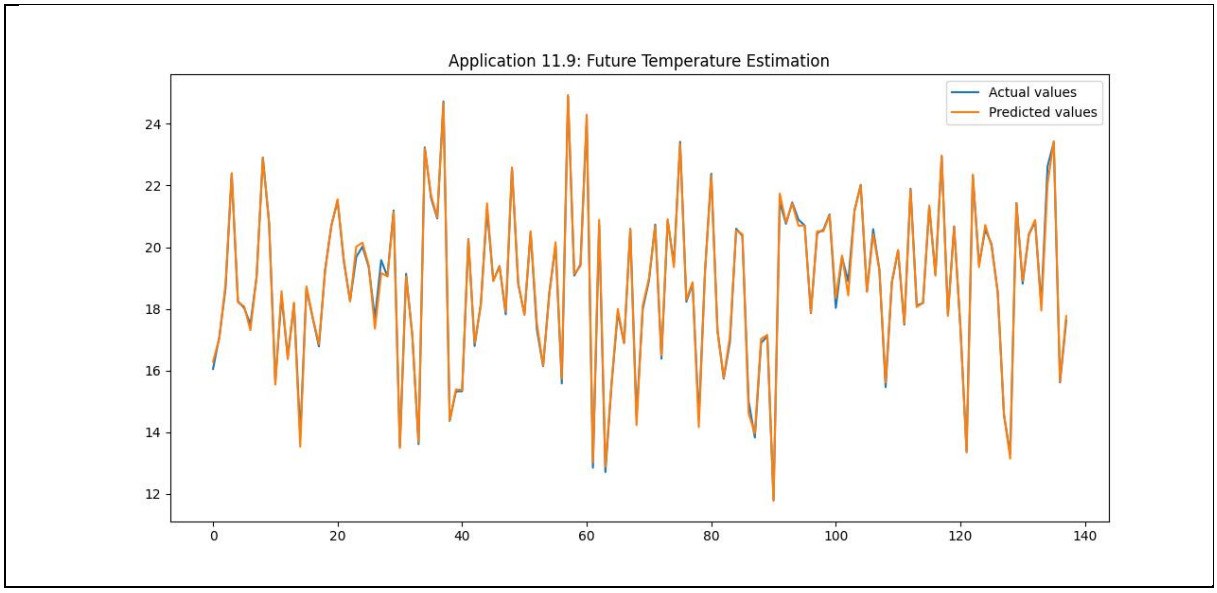
network using stochastic gradient descent to minimize mean absolute error over 3000 epochs. The final results are evaluated by comparing predicted versus actual values on a test set plot, and the trained model is saved for future temperature estimations.

Code:

```
import os
import pandas as pd
import numpy as np
import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from matplotlib import pyplot as plt

# --- UPDATED PATH CONFIGURATION ---
# Points directly to the 'data' and 'Models' folders in your current directory
current_dir = os.path.dirname(os.path.abspath(__file__))
TEMPERATURE_DATA_PATH = os.path.join(current_dir, "data", "temperature_dataset.csv")
KERAS_MODEL_DIR = os.path.join(current_dir, "Models")

if not os.path.exists(KERAS_MODEL_DIR):
    os.makedirs(KERAS_MODEL_DIR)

# 1. Load the Filtered CSV Data
df = pd.read_csv(TEMPERATURE_DATA_PATH)

# Use the Room_Temp column. [::4] downsamples from 15-min to 1-hour intervals
y = df['Room_Temp'][::4]
prev_values_count = 5

# 2. Windowing Logic (Time-Series Features)
X = pd.DataFrame()
for i in range(prev_values_count, 0, -1):
    X['t-' + str(i)] = y.shift(i)

# Remove the empty rows created by shifting
X = X[prev_values_count:]
y = y[prev_values_count:]

# 3. Data Split and Normalization
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

train_mean = X_train.mean()
train_std = X_train.std()

X_train = (X_train - train_mean) / train_std
X_test = (X_test - train_mean) / train_std

# 4. Model Definition (Application 11.9)
model = keras.models.Sequential([
    keras.layers.Dense(100, input_shape=[5], activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(1) # Final estimation output
])

# 5. Compilation and Training
model.compile(optimizer=keras.optimizers.SGD(learning_rate=5e-4),
              loss=keras.losses.MeanAbsoluteError())

print("Starting training...")
model.fit(X_train,
          y_train,
          batch_size=128,
          epochs=3000,
          verbose=1)
```

```
# 6. Prediction and Visualization
y_train_predicted = model.predict(X_train)
y_test_predict = model.predict(X_test)

fig, ax = plt.subplots(1, 1, figsize=(10, 5))
ax.plot(y_test.to_numpy(), label="Actual values")
ax.plot(y_test_predict, label="Predicted values")
plt.title("Application 11.9: Future Temperature Estimation")
plt.legend()
plt.show()

# 7. Metrics
# Calculating the root of MAE as per your original request
mae_train = np.sqrt(mean_absolute_error(y_train, y_train_predicted))
mae_test = np.sqrt(mean_absolute_error(y_test, y_test_predict))

print(f"¥nTraining set MAE (Sqrt): {mae_train:.4f}")
print(f"Test set MAE (Sqrt): {mae_test:.4f}")

# 8. Save Model
model_save_path = os.path.join(KERAS_MODEL_DIR, "temperature_pred_mlp.h5")
model.save(model_save_path)
print(f"Model saved to: {model_save_path}")
```

Results:



Bonus:

Listing10_4.py Code:

This code script visualizes the activation surface of a single neuron by passing a 2D grid of data points through a neural network consisting of a single dense layer with a sigmoid activation function. After manually initializing the neuron's weights and bias, the code generates a coordinate grid ranging from -5 to 5, calculates the inference for all 10,000 points, and reshapes the resulting predictions to create a 3D surface plot. The final visualization

illustrates how the neuron divides the input space into a gradient of values between 0 and 1, demonstrating the fundamental "S-shaped" curve of the sigmoid function in three dimensions.

Code:

```
import os
import sys

# --- 1. Suppress Warnings ---
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import warnings
warnings.filterwarnings("ignore")

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

def main():
    print("Initializing Model...")

    # 1. Define Weights and Biases
    # We want 1 neuron with 2 inputs.
    # Weights shape must be (Input_Dim, Units) -> (2, 1)
    # This ensures we get exactly 1 output per sample.
    w_init = tf.constant_initializer([[0.5], [-0.5]])
    b_init = tf.constant_initializer([0.0])

    # 2. Create Model
    # Using an explicit Input layer fixes the 'UserWarning'
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(2,)), # Input has 2 features (x and y)
        tf.keras.layers.Dense(
            units=1,                       # ONE output (important for plotting)
            kernel_initializer=w_init,
            bias_initializer=b_init,
            activation='sigmoid'
        )
    ])

    # 3. Generate Data Grid
    # Range -5 to 5 with step 0.1 = 100 points per axis
    # 100 * 100 = 10,000 total data points
    x_range = np.arange(-5, 5, 0.1)
    y_range = np.arange(-5, 5, 0.1)
    x_grid, y_grid = np.meshgrid(x_range, y_range)

    # Flatten data to shape (10000, 2) for the model
    x_flat = x_grid.ravel()
    y_flat = y_grid.ravel()
    input_data = np.column_stack((x_flat, y_flat))

    # 4. Run Inference
    print(f"Calculating output for {input_data.shape[0]} points...")
    Z = model.predict(input_data, verbose=0)

    # 5. Reshape for Plotting
    # Z comes out as shape (10000, 1). We flatten it to (10000,)
    # so it can be reshaped to (100, 100)
    Z = Z.flatten()

    if Z.size != x_grid.size:
        print(f"Error: Model produced {Z.size} outputs, but grid needs {x_grid.size}.")
        print("Check if 'units' in Dense layer is set to 1.")
        return
```

```
    Z_reshaped = Z.reshape(x_grid.shape)

    # 6. Plot
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"}, figsize=(10, 8))
    surf = ax.plot_surface(x_grid, y_grid, Z_reshaped, cmap=cm.coolwarm, linewidth=0, antialiased=False)

    ax.set_xlabel('Input X')
    ax.set_ylabel('Input Y')
    ax.set_zlabel('Sigmoid Output')
    ax.set_title('Single Neuron Activation')
    fig.colorbar(surf, shrink=0.5, aspect=5)

    print("Displaying plot...")
    plt.show()

if __name__ == "__main__":
    main()
```

Results: