Name: Mehar Fatima

Roll Number: 241030051

Kaggle User ID: MeharFatima01

```
# Importing libraries
from tensorflow.keras.preprocessing.sequence import pad_sequences
import pandas as pd
# For handling data
import numpy as np
# For numerical operations
from sklearn.preprocessing import StandardScaler, LabelEncoder
# Standardizes feature values to have mean 0 and variance 1
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
BatchNormalization, LeakyReLU
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau          # Helps in optimizing training and
preventing overfitting
from tensorflow.keras.regularizers import l2
# Adds L2 regularization to prevent overfitting
from scipy.stats import skew, kurtosis, entropy
from imblearn.over_sampling import SMOTE
```

1. Dataset reading

read_data -

This function reads data from a file and processes it into numerical sequences (features) and labels (if applicable). data: A list that will store numerical feature values. labels: A list to store labels (only if is_train=True). strip() removes any leading/trailing whitespace. split(',') splits each line by commas to extract individual values. Labels are stored separately

Padding-

is done to ensure all input sequences have same length by adding zeros to shorter sequences.

Encoding-

ML models cannot directly work with categorical labels so this converts class labels into numerical values.

```
# Reading and Preprocessing Data
def read_data(filename, is_train=True):
    data = []
    labels = [] if is_train else None
    with open(filename, 'r') as f:
        for line in f:
            parts = line.strip().split(',')
            if is_train:
                labels.append(parts[1])
                values = [float(x) for x in parts[2:]]
            else:
                values = [float(x) for x in parts[1:]]
            data.append(values)
    return data, labels

# Loading data
train_data, train_labels = read_data("train.txt", is_train=True)
test_data, _ = read_data("test.txt", is_train=False)

# Converting to Pandas DataFrames
train_df = pd.DataFrame(train_data)
test_df = pd.DataFrame(test_data)

# Padding sequences
max_len = max(len(seq) for seq in train_data + test_data)
train_data_padded = pad_sequences(train_data, maxlen=max_len,
padding='post', dtype='float32', value=0)
test_data_padded = pad_sequences(test_data, maxlen=max_len,
padding='post', dtype='float32', value=0)

# Encoding categorical labels into numeric values
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
num_classes = len(label_encoder.classes_)
```

2. Features generation

Choosing these features to ensures both time-domain and frequency-domain properties are captured.

Reason for chosing these features:-

1)Mean-Captures the overall signal strength. 2)Min & Max-Helps identify the signal range. 3)Median-A robust measure of central tendency, less affected by outliers. 4)Standard Deviation-Measures dispersion, useful for identifying variability. 5)Skewness-Indicates the asymmetry of the distribution. 6)Kurtosis-Measures the peak of the distribution, helpful in detecting anomalies. 7)Peak-to-Peak (ptp)-The difference between the maximum and minimum value, useful for amplitude analysis. 8)Mean Absolute Difference-Captures fluctuations in the sequence. 9)Mean Square Difference-Useful for analyzing the variation between points. 10)FFT Mean & Variance-Extracts frequency domain information. 11)RMS (Root Mean Square)-Indicates the energy content of the signal. 12)Entropy-Measures randomness or disorder in the signal. 13)Signal Energy-Quantifies the total power of the signal.

Feature Scaling

Done to ensure that all features contribute equally to the learning process. StandardScaler() transforms data to have a mean of 0 and variance of 1 preventing large-valued features from dominating small-valued ones.

Handling Class Imbalance

The dataset is imbalanced, the model may be biased toward the majority class. SMOTE (Synthetic Minority Over-sampling Technique) generates synthetic samples for the minority class.

```python
# 2. Feature Engineering
def extract_features(data):
    skew_values = skew(data, axis=1)
    kurtosis_values = kurtosis(data, axis=1)
    rms_values = np.sqrt(np.mean(np.square(data), axis=1))
    entropy_values = np.apply_along_axis(lambda x: entropy(np.abs(x) +
1e-10), axis=1, arr=data)
    signal_energy = np.sum(np.square(data), axis=1)
    mean_abs_dev = np.mean(np.abs(data - np.mean(data, axis=1,
keepdims=True)), axis=1)
    return np.column_stack([
        np.mean(data, axis=1),
        np.min(data, axis=1),
        np.max(data, axis=1),
        np.median(data, axis=1),
        np.std(data, axis=1),
        skew_values,
        kurtosis_values,
        np.ptp(data, axis=1),
        np.mean(np.abs(np.diff(data, axis=1)), axis=1),
        np.mean(np.square(np.diff(data, axis=1)), axis=1),
        np.fft.fft(data).real.mean(axis=1),
        np.fft.fft(data).real.var(axis=1),
        rms_values,
        entropy_values,
        signal_energy,
        mean_abs_dev
    ])

train_features = extract_features(train_data_padded)
test_features = extract_features(test_data_padded)

# Standardize Features
scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)

# Handling Class Imbalance by oversampling minority class examples
smote = SMOTE(random_state=42)
train_features, train_labels_encoded =
smote.fit_resample(train_features, train_labels_encoded)
```

3. Model selection and training

Model choice

ANN is chosen over LR due to the complexity and nature of the dataset. It is better for High-Dimensional, Non-Linear Data. It Can Learn Complex Representations. It is more Robust to Feature Engineering & imbalanced data.

More specifics

L2 Regularization: Prevents overfitting by adding weight decay. Batch Normalization: Stabilizes training and improves convergence. LeakyReLU used instead of ReLU: Avoids "dying neurons" problem, where neurons become inactive. Dropout (0.3): Prevents overfitting by randomly deactivating neurons. ReduceLROnPlateau-Reduces the learning rate if validation loss stops improving. EarlyStopping-Stops training when the model stops improving to prevent overfitting.

```
#3. Model Training (Using ANN)
X_train, X_val, y_train, y_val = train_test_split(train_features,
train_labels_encoded, test_size=0.2, random_state=42)

ann_model = Sequential([
    Dense(512, input_shape=(train_features.shape[1],),
kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(),
    Dropout(0.3),

    Dense(256, kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(),
    Dropout(0.3),

    Dense(128, kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(),
    Dropout(0.3),

    Dense(64, kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(),
    Dropout(0.3),

    Dense(num_classes, activation='softmax')
])

ann_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Learning rate scheduler
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=5, min_lr=1e-6)
early_stopping = EarlyStopping(monitor='val_loss', patience=15,
restore_best_weights=True)

ann_model.fit(X_train, y_train, validation_data=(X_val, y_val),
epochs=300, batch_size=128, callbacks=[early_stopping, reduce_lr],
verbose=1)

# Evaluate Model
y_pred_ann = np.argmax(ann_model.predict(X_val), axis=-1)
ann_f1 = f1_score(y_val, y_pred_ann, average='macro')
print("ANN Model F1-score:", ann_f1)
```

```
C:\Users\mfati\anaconda3\Lib\site-packages\keras\src\layers\core\
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.5637 - loss:
1.6257 - val_accuracy: 0.7429 - val_loss: 1.2669 - learning_rate:
0.0010
Epoch 2/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.8207 - loss:
1.0676 - val_accuracy: 0.7714 - val_loss: 1.1648 - learning_rate:
0.0010
Epoch 3/300
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.8014 - loss:
1.0721 - val_accuracy: 0.7714 - val_loss: 1.1173 - learning_rate:
0.0010
Epoch 4/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.8267 - loss:
0.9928 - val_accuracy: 0.7755 - val_loss: 1.0923 - learning_rate:
0.0010
Epoch 5/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.8203 - loss:
0.9822 - val_accuracy: 0.7796 - val_loss: 1.0670 - learning_rate:
0.0010
Epoch 6/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.8263 - loss:
0.9496 - val_accuracy: 0.8122 - val_loss: 1.0371 - learning_rate:
0.0010
Epoch 7/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.8390 - loss:
0.9388 - val_accuracy: 0.7959 - val_loss: 1.0259 - learning_rate:
0.0010
Epoch 8/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.8465 - loss:
0.9071 - val_accuracy: 0.8041 - val_loss: 1.0133 - learning_rate:
0.0010
Epoch 9/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.8391 - loss:
0.9121 - val_accuracy: 0.8163 - val_loss: 1.0021 - learning_rate:
0.0010
Epoch 10/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.8317 - loss:
0.9124 - val_accuracy: 0.8286 - val_loss: 0.9984 - learning_rate:
0.0010
Epoch 11/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.8746 - loss:
0.8490 - val_accuracy: 0.8204 - val_loss: 0.9804 - learning_rate:
0.0010
Epoch 12/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - accuracy: 0.8607 - loss:
0.8391 - val_accuracy: 0.8327 - val_loss: 0.9737 - learning_rate:
0.0010
Epoch 13/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.8636 - loss:
0.8386 - val_accuracy: 0.7959 - val_loss: 0.9851 - learning_rate:
0.0010
Epoch 14/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.8454 - loss:
0.8512 - val_accuracy: 0.8041 - val_loss: 0.9848 - learning_rate:
0.0010
Epoch 15/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.8566 - loss:
```

```
0.8316 - val_accuracy: 0.8122 - val_loss: 0.9658 - learning_rate:
0.0010
Epoch 16/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.8528 - loss:
0.8371 - val_accuracy: 0.8286 - val_loss: 0.9464 - learning_rate:
0.0010
Epoch 17/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.8446 - loss:
0.8282 - val_accuracy: 0.8082 - val_loss: 0.9434 - learning_rate:
0.0010
Epoch 18/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.8616 - loss:
0.7811 - val_accuracy: 0.8041 - val_loss: 0.9367 - learning_rate:
0.0010
Epoch 19/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.8506 - loss:
0.7782 - val_accuracy: 0.8327 - val_loss: 0.9029 - learning_rate:
0.0010
Epoch 20/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.8807 - loss:
0.7265 - val_accuracy: 0.7878 - val_loss: 0.9095 - learning_rate:
0.0010
Epoch 21/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.8773 - loss:
0.7241 - val_accuracy: 0.8082 - val_loss: 0.8942 - learning_rate:
0.0010
Epoch 22/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.8612 - loss:
0.7446 - val_accuracy: 0.8327 - val_loss: 0.8816 - learning_rate:
0.0010
Epoch 23/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.8577 - loss:
0.7345 - val_accuracy: 0.7918 - val_loss: 0.8911 - learning_rate:
0.0010
Epoch 24/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.8553 - loss:
0.7249 - val_accuracy: 0.7755 - val_loss: 0.8867 - learning_rate:
0.0010
Epoch 25/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.8842 - loss:
0.6773 - val_accuracy: 0.7878 - val_loss: 0.8711 - learning_rate:
0.0010
Epoch 26/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.8675 - loss:
0.6906 - val_accuracy: 0.8082 - val_loss: 0.8559 - learning_rate:
0.0010
Epoch 27/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8732 - loss:
0.6727 - val_accuracy: 0.8163 - val_loss: 0.8417 - learning_rate:
```

```
0.0010
Epoch 28/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8761 - loss: 
0.6721 - val_accuracy: 0.7796 - val_loss: 0.8944 - learning_rate: 
0.0010
Epoch 29/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.8707 - loss: 
0.6701 - val_accuracy: 0.8286 - val_loss: 0.8087 - learning_rate: 
0.0010
Epoch 30/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.8823 - loss: 
0.6512 - val_accuracy: 0.8000 - val_loss: 0.8040 - learning_rate: 
0.0010
Epoch 31/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.8612 - loss: 
0.6693 - val_accuracy: 0.8163 - val_loss: 0.7969 - learning_rate: 
0.0010
Epoch 32/300
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.8715 - loss: 
0.6415
```

4. Prediction for test data

```python
# 4. Prediction on Test Set
test_predictions_encoded = np.argmax(ann_model.predict(test_features), 
axis=-1)
test_predictions_labels = 
label_encoder.inverse_transform(test_predictions_encoded)

# Create submission file
submission_df = pd.DataFrame({'index': 
range(len(test_predictions_labels)), 'label': 
test_predictions_labels})
submission_df.to_csv('submission.csv', index=False)
print("Submission file generated successfully!")
```

Collaboration and Acknowledgment Statement

In completing this assignment, I referred to multiple resources for better understanding. I used Google to look up Python syntax and built-in functions whenever needed. To understand the dataset and its context, I reviewed the following research papers: [1]Souza V.M.A. Asphalt pavement classification using smartphone accelerometer and Complexity Invariant Distance. Engineering Applications of Artificial Intelligence, Volume 74, pp. 198-211. [2]Souza V.M.A., Cherman E.A., Rossi R.G., Souza R.A. Towards automatic evaluation of asphalt irregularity using smartphone sensors. International Symposium on Intelligent Data Analysis (2017), pp. 322-333. For understanding some python syntax I also referred to few youtube videos. Additionally, I took some help from ChatGPT for understanding the implementation of ANN models, particularly regarding the choice of parameters in Python. However, the final code and approach were developed independently with understanding.