Mukundi Mangena and Ntsako Phiri

Model preprocessed using windowing

Download the dataset and prepare it for preprocessing.

In [1]:
```
import numpy as np
import pandas as pd
from pandas import DataFrame
import pandas as pd

def get_ds_infos():
    """
    Read the file includes data subject information.

    Data Columns:
    0: code [1-24]
    1: weight [kg]
    2: height [cm]
    3: age [years]
    4: gender [0:Female, 1:Male]

    Returns:
        A pandas DataFrame that contains inforamtion about data subjects' attrib
    """

    dss = pd.read_csv("data_subjects_info.csv")
    print("[INFO] -- Data subjects' information is imported.")

    return dss

def set_data_types(data_types=["userAcceleration"]):
    """
    Select the sensors and the mode to shape the final dataset.

    Args:
        data_types: A list of sensor data type from this list: [attitude, gravit

    Returns:
        It returns a list of columns to use for creating time-series from files.
    """
    dt_list = []
    for t in data_types:
        if t != "attitude":
            dt_list.append([t+".x",t+".y",t+".z"])
        else:
            dt_list.append([t+".roll", t+".pitch", t+".yaw"])

    return dt_list


def creat_time_series(dt_list, act_labels, trial_codes, mode="mag", labeled=True
    """
    Args:
```

```
        dt_list: A list of columns that shows the type of data we want.
        act_labels: list of activites
        trial_codes: list of trials
        mode: It can be "raw" which means you want raw data
        for every dimention of each data type,
        [attitude(roll, pitch, yaw); gravity(x, y, z); rotationRate(x, y, z); us
        or it can be "mag" which means you only want the magnitude for each data
        labeled: True, if we want a labeld dataset. False, if we only want senso

    Returns:
        It returns a time-series of sensor data.

    """
    num_data_cols = len(dt_list) if mode == "mag" else len(dt_list*3)

    if labeled:
        dataset = np.zeros((0,num_data_cols+7)) # "7" --> [act, code, weight, he
    else:
        dataset = np.zeros((0,num_data_cols))

    ds_list = get_ds_infos()

    print("[INFO] -- Creating Time-Series")
    for sub_id in ds_list["code"]:
        for act_id, act in enumerate(act_labels):
            for trial in trial_codes[act_id]:
                fname = 'A_DeviceMotion_data/'+act+'_'+str(trial)+'/sub_'+str(in
                raw_data = pd.read_csv(fname)
                raw_data = raw_data.drop(['Unnamed: 0'], axis=1)
                vals = np.zeros((len(raw_data), num_data_cols))
                for x_id, axes in enumerate(dt_list):
                    if mode == "mag":
                        vals[:,x_id] = (raw_data[axes]**2).sum(axis=1)**0.5
                    else:
                        vals[:,x_id*3:(x_id+1)*3] = raw_data[axes].values
                    vals = vals[:,:num_data_cols]
                if labeled:
                    lbls = np.array([[act_id,
                            sub_id-1,
                            ds_list["weight"][sub_id-1],
                            ds_list["height"][sub_id-1],
                            ds_list["age"][sub_id-1],
                            ds_list["gender"][sub_id-1],
                            trial
                           ]]*len(raw_data))
                    vals = np.concatenate((vals, lbls), axis=1)
                dataset = np.append(dataset,vals, axis=0)
    cols = []
    for axes in dt_list:
        if mode == "raw":
            cols += axes
        else:
            cols += [str(axes[0][:-2])]

    if labeled:
        cols += ["act", "id", "weight", "height", "age", "gender", "trial"]

    dataset = pd.DataFrame(data=dataset, columns=cols)
    return dataset
#_______________________________
```

```
ACT_LABELS = ["dws","ups", "wlk", "jog", "std", "sit"]
TRIAL_CODES = {
    ACT_LABELS[0]:[1,2,11],
    ACT_LABELS[1]:[3,4,12],
    ACT_LABELS[2]:[7,8,15],
    ACT_LABELS[3]:[9,16],
    ACT_LABELS[4]:[6,14],
    ACT_LABELS[5]:[5,13]
}

## Here we set parameter to build labeld time-series from dataset of "(A)DeviceM
## attitude(roll, pitch, yaw); gravity(x, y, z); rotationRate(x, y, z); userAcce
sdt = ["attitude", "userAcceleration","rotationRate"]
print("[INFO] -- Selected sensor data types: "+str(sdt))
act_labels = ACT_LABELS [0:6] # ["dws","ups", "wlk", "jog", "std"]
print("[INFO] -- Selected activites: "+str(act_labels))
trial_codes = [TRIAL_CODES[act] for act in act_labels]
dt_list = set_data_types(sdt)
dataset = creat_time_series(dt_list, act_labels, trial_codes, mode="raw", labele
print("[INFO] -- Shape of time-Series dataset:"+str(dataset.shape))
dataset.head()
```

```
[INFO] -- Selected sensor data types: ['attitude', 'userAcceleration', 'rotationR
ate']
[INFO] -- Selected activites: ['dws', 'ups', 'wlk', 'jog', 'std', 'sit']
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[1], line 129
    127 trial_codes = [TRIAL_CODES[act] for act in act_labels]
    128 dt_list = set_data_types(sdt)
--> 129 dataset = creat_time_series(dt_list, act_labels, trial_codes, mode=     ,
labeled=True)
    130 print("[INFO] -- Shape of time-Series dataset:"+str(dataset.shape))
    131 dataset.head()

Cell In[1], line 69, in creat_time_series(dt_list, act_labels, trial_codes, mode,
labeled)
     66 else:
     67     dataset = np.zeros((0,num_data_cols))
---> 69 ds_list = get_ds_infos()
     71 print("[INFO] -- Creating Time-Series")
     72 for sub_id in ds_list["code"]:

Cell In[1], line 21, in get_ds_infos()
      6 def get_ds_infos():
      7     """
      8     Read the file includes data subject information.
      9
   (...)     18         A pandas DataFrame that contains inforamtion about data s
ubjects' attributes
     19     """
---> 21     dss = pd.read_csv(                          )
     22     print("[INFO] -- Data subjects' information is imported.")
     24     return dss

File c:\Users\btmuk\OneDrive\Desktop\2025\DSP\.venv\Lib\site-packages\pandas\io\p
arsers\readers.py:1026, in read_csv(filepath_or_buffer, sep, delimiter, header, n
ames, index_col, usecols, dtype, engine, converters, true_values, false_values, s
kipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filt
er, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col,
date_parser, date_format, dayfirst, cache_dates, iterator, chunksize, compressio
n, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapecha
r, comment, encoding, encoding_errors, dialect, on_bad_lines, delim_whitespace, l
ow_memory, memory_map, float_precision, storage_options, dtype_backend)
   1013 kwds_defaults = _refine_defaults_read(
   1014     dialect,
   1015     delimiter,
   (...)   1022     dtype_backend=dtype_backend,
   1023 )
   1024 kwds.update(kwds_defaults)
-> 1026 return _read(filepath_or_buffer, kwds)

File c:\Users\btmuk\OneDrive\Desktop\2025\DSP\.venv\Lib\site-packages\pandas\io\p
arsers\readers.py:620, in _read(filepath_or_buffer, kwds)
    617 _validate_names(kwds.get("names", None))
    619 # Create the parser.
--> 620 parser = TextFileReader(filepath_or_buffer, **kwds)
    622 if chunksize or iterator:
    623     return parser

File c:\Users\btmuk\OneDrive\Desktop\2025\DSP\.venv\Lib\site-packages\pandas\io\p
arsers\readers.py:1620, in TextFileReader.__init__(self, f, engine, **kwds)
   1617     self.options["has_index_names"] = kwds["has_index_names"]
   1619 self.handles: IOHandles | None = None
-> 1620 self._engine = self._make_engine(f, self.engine)
```

```
File c:\Users\btmuk\OneDrive\Desktop\2025\DSP\.venv\Lib\site-packages\pandas\io\parsers\readers.py:1880, in TextFileReader._make_engine(self, f, engine)
   1878     if "b" not in mode:
   1879         mode += "b"
-> 1880 self.handles = get_handle(
   1881     f,
   1882     mode,
   1883     encoding=self.options.get(          , None),
   1884     compression=self.options.get(             , None),
   1885     memory_map=self.options.get(            , False),
   1886     is_text=is_text,
   1887     errors=self.options.get(                ,         ),
   1888     storage_options=self.options.get(                , None),
   1889 )
   1890 assert self.handles is not None
   1891 f = self.handles.handle

File c:\Users\btmuk\OneDrive\Desktop\2025\DSP\.venv\Lib\site-packages\pandas\io\common.py:873, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    868 elif isinstance(handle, str):
    869     # Check whether the filename is to be opened in binary mode.
    870     # Binary mode does not support 'encoding' and 'newline'.
    871     if ioargs.encoding and "b" not in ioargs.mode:
    872         # Encoding
--> 873         handle = open(
    874             handle,
    875             ioargs.mode,
    876             encoding=ioargs.encoding,
    877             errors=errors,
    878             newline=  ,
    879         )
    880     else:
    881         # Binary mode
    882         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'data_subjects_info.csv'
```

drop the redudant tables

In [2]:
```
dataset = dataset.drop(columns=["id", "weight", "height", "age", "gender", "tria
dataset.head()
```

Out[2]:

	attitude.roll	attitude.pitch	attitude.yaw	userAcceleration.x	userAcceleration.y	user
0	1.528132	-0.733896	0.696372	0.294894	-0.184493	
1	1.527992	-0.716987	0.677762	0.219405	0.035846	
2	1.527765	-0.706999	0.670951	0.010714	0.134701	
3	1.516768	-0.704678	0.675735	-0.008389	0.136788	
4	1.493941	-0.703918	0.672994	0.199441	0.353996	

Note that this database has all the data in the dataset it will require to be trimmed to alow for more effective models.

Data Pre-Processing Steps

Setting up the input and output data

In [3]:
```
import sys
print(sys.executable)
print(dataset.shape)
```

```
c:\Users\btmuk\OneDrive\Desktop\2025\DSP\.venv\Scripts\python.exe
(1412865, 10)
```

In [4]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

x  = dataset[dataset.columns[:-1]].values
y = dataset["act"].values
print(x[:10])
print(y[:10])
print("[INFO] -- Dataset is split into input and output data.")
```

```
[[ 1.528132 -0.733896  0.696372  0.294894 -0.184493  0.377542  0.316738
   0.77818   1.082764]
 [ 1.527992 -0.716987  0.677762  0.219405  0.035846  0.114866  0.842032
   0.424446  0.643574]
 [ 1.527765 -0.706999  0.670951  0.010714  0.134701 -0.167808 -0.138143
  -0.040741  0.343563]
 [ 1.516768 -0.704678  0.675735 -0.008389  0.136788  0.094958 -0.025005
  -1.048717  0.03586 ]
 [ 1.493941 -0.703918  0.672994  0.199441  0.353996 -0.044299  0.114253
  -0.91289   0.047341]
 [ 1.476302 -0.700807  0.669443  0.168241  0.145906  0.012455  0.187742
  -0.763656  0.226057]
 [ 1.455153 -0.694408  0.662593  0.079382 -0.026344 -0.19559   0.343096
  -0.80382   0.278468]
 [ 1.441702 -0.69071   0.656459  0.06936   0.072678 -0.10292   0.176202
  -0.172756  0.056415]
 [ 1.44344  -0.691905  0.651196  0.072889  0.079921 -0.075323  0.274786
   0.446585 -0.132766]
 [ 1.443071 -0.693039  0.638198  0.098347 -0.017021 -0.19731   0.633672
   0.316372 -0.115137]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[INFO] -- Dataset is split into input and output data.
```

Encode the Labels

To make them compatible with Keras

In [5]:
```
le = LabelEncoder()
y = le.fit_transform(y)
y= to_categorical(y)
print("[INFO] -- Labels are encoded and one-hot encoded.")
```

```
[INFO] -- Labels are encoded and one-hot encoded.
```

Preprocessed by equal Windows The Data

Since the data is time series data when implementing a CNN or RNN

In [6]:
```
def create_segments(data, labels, window_size=50, step=25):
    x, y = [], []
    for start in range(0, len(data) - window_size, step):
        end = start + window_size
        x.append(data[start:end])
        y.append(labels[end - 1])  # label of the last sample in window
    return np.array(x), np.array(y)

x_segments, y_segments = create_segments(x, y)
print(x_segments[:5])
print(y_segments.shape)
```

```
[[[ 1.528132 -0.733896  0.696372 ...  0.316738  0.77818   1.082764]
  [ 1.527992 -0.716987  0.677762 ...  0.842032  0.424446  0.643574]
  [ 1.527765 -0.706999  0.670951 ... -0.138143 -0.040741  0.343563]
  ...
  [ 1.617452 -0.803518  0.920612 ... -0.525354 -0.589796  0.004023]
  [ 1.61021  -0.803335  0.931939 ... -0.333441 -0.891791 -0.031637]
  [ 1.598267 -0.804383  0.941282 ... -0.304708 -0.941823 -0.065555]]

 [[ 1.231014 -0.786512  0.649841 ... -2.378386  3.535351 -1.148708]
  [ 1.347834 -0.829441  0.717322 ... -3.073846  3.196258 -1.527009]
  [ 1.446529 -0.862922  0.799754 ... -2.689446  0.650326 -0.814967]
  ...
  [ 1.256518 -0.827858  1.228694 ... -0.016013 -0.919413  1.007832]
  [ 1.256213 -0.809375  1.242856 ... -0.150397 -0.264595  0.939824]
  [ 1.255842 -0.80108   1.240967 ...  0.416105  0.185818 -0.033151]]

 [[ 1.583899 -0.805282  0.946705 ... -0.119064 -0.926548 -0.050391]
  [ 1.565382 -0.806011  0.949998 ... -0.140231 -1.085363 -0.009743]
  [ 1.548804 -0.80442   0.956486 ... -0.27648  -1.122698  0.120744]
  ...
  [ 1.669818 -0.839965  1.816442 ... -0.372399 -0.314253 -0.280073]
  [ 1.668462 -0.843037  1.82705  ... -0.308166 -0.581435 -0.120117]
  [ 1.657153 -0.842865  1.83408  ... -0.178372 -0.981994  0.037264]]

 [[ 1.24478  -0.806294  1.218805 ...  0.761158  0.368247 -0.66443 ]
  [ 1.243234 -0.810727  1.197761 ...  0.359515  0.787237 -0.13473 ]
  [ 1.266007 -0.801499  1.206465 ... -0.376665  1.009725  0.807356]
  ...
  [ 0.899696 -0.747679  2.212082 ... -1.980115  0.189763 -0.518479]
  [ 0.928464 -0.781327  2.247116 ... -2.038452  0.206186 -0.578728]
  [ 0.954615 -0.813852  2.284163 ... -1.964891 -0.222545 -0.51998 ]]

 [[ 1.63249  -0.840201  1.834923 ...  0.065177 -1.467124  0.19982 ]
  [ 1.588541 -0.83389   1.822689 ...  0.691736 -1.883562  0.434398]
  [ 1.512851 -0.820085  1.779413 ...  1.972599 -2.365554  0.777247]
  ...
  [ 1.436506 -0.996786 -3.122631 ... -0.925317 -0.198186 -0.585389]
  [ 1.453074 -1.009788 -3.091882 ... -0.876384 -0.61418  -0.493715]
  [ 1.461599 -1.019192 -3.064953 ... -0.666445 -0.737028 -0.324381]]]
(56513, 6)
```

Setting up the Data on holdout Methods and Train Test Split

In [7]:
```
print("[INFO] --  Train Test APlit.")


# Step 1: Split into train (70%) and temp (30%)
```

```
X_train, X_temp, y_train, y_temp = train_test_split(  x_segments, y_segments, te

# Step 2: Split temp into validation (60%) and test (40%)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, r

print(f"Train: {len(X_train)}, Validation: {len(X_val)}, Test: {len(X_test)}")
print(y_train.shape[0])
```

```
[INFO] --  Train Test APlit.
Train: 39559, Validation: 8477, Test: 8477
39559
```

Building the Model

In [8]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropou
#creating a sequential model
model = Sequential()
# First layer: Conv1D with 64 filters, kernel size of 3, ReLU activation
model.add(Conv1D(64, 3, activation='relu', input_shape=(X_train.shape[1], X_trai
# Second layer: Conv1D with 128 filters, kernel size of 3, ReLU activation
model.add(MaxPooling1D(2))
#randomly dropping 50% of the neurons to prevent overfitting
model.add(Dropout(0.5))
#Converts pooled feature maps to a 1D feature vector
model.add(Flatten())
# Fully connected layer with 100 neurons and ReLU activation
model.add(Dense(100, activation='relu'))
#Number of activity classes is 6, using softmax activation for multi-class class
model.add(Dense(y_train.shape[1], activation='softmax'))
# Compiling the model with Adam optimizer, categorical crossentropy loss, and ac

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accur
```

```
c:\Users\btmuk\OneDrive\Desktop\2025\DSP\.venv\Lib\site-packages\keras\src\layers
\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input
_dim` argument to a layer. When using Sequential models, prefer using an `Input(s
hape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Training The Model

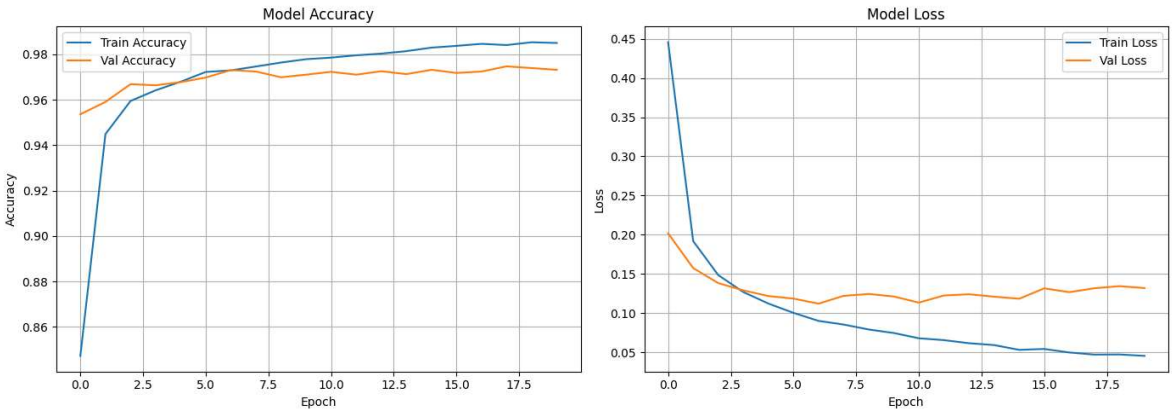
In [9]:
```
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

```
Epoch 1/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - accuracy: 0.7397 - loss: 0.7244 - val
_accuracy: 0.9536 - val_loss: 0.2017
Epoch 2/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 8s 8ms/step - accuracy: 0.9405 - loss: 0.2003 - val_
accuracy: 0.9590 - val_loss: 0.1574
Epoch 3/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9592 - loss: 0.1522 - val_
accuracy: 0.9669 - val_loss: 0.1382
Epoch 4/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9660 - loss: 0.1221 - val_
accuracy: 0.9664 - val_loss: 0.1289
Epoch 5/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9698 - loss: 0.1034 - val_
accuracy: 0.9678 - val_loss: 0.1217
Epoch 6/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9733 - loss: 0.0975 - val_
accuracy: 0.9698 - val_loss: 0.1184
Epoch 7/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.9729 - loss: 0.0878 - val_
accuracy: 0.9731 - val_loss: 0.1120
Epoch 8/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 8s 8ms/step - accuracy: 0.9754 - loss: 0.0849 - val_
accuracy: 0.9724 - val_loss: 0.1219
Epoch 9/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.9765 - loss: 0.0786 - val_
accuracy: 0.9699 - val_loss: 0.1243
Epoch 10/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.9788 - loss: 0.0706 - val_
accuracy: 0.9711 - val_loss: 0.1211
Epoch 11/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9786 - loss: 0.0671 - val_
accuracy: 0.9723 - val_loss: 0.1133
Epoch 12/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9814 - loss: 0.0605 - val_
accuracy: 0.9711 - val_loss: 0.1224
Epoch 13/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9806 - loss: 0.0618 - val_
accuracy: 0.9726 - val_loss: 0.1239
Epoch 14/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9825 - loss: 0.0544 - val_
accuracy: 0.9713 - val_loss: 0.1209
Epoch 15/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9835 - loss: 0.0501 - val_
accuracy: 0.9732 - val_loss: 0.1182
Epoch 16/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.9846 - loss: 0.0488 - val_
accuracy: 0.9718 - val_loss: 0.1315
Epoch 17/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - accuracy: 0.9845 - loss: 0.0478 - va
l_accuracy: 0.9724 - val_loss: 0.1266
Epoch 18/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9849 - loss: 0.0448 - val_
accuracy: 0.9747 - val_loss: 0.1316
Epoch 19/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9862 - loss: 0.0454 - val_
accuracy: 0.9740 - val_loss: 0.1343
Epoch 20/20
989/989 ━━━━━━━━━━━━━━━━━━━━ 10s 10ms/step - accuracy: 0.9853 - loss: 0.0427 - va
l_accuracy: 0.9732 - val_loss: 0.1318
```

Out[9]: <keras.src.callbacks.history.History at 0x1b7d4b92d10>

Plots training and validation accuracy and loss curves.

In [10]:
```
import matplotlib.pyplot as plt

def plot_training_history(history):
    """
    Plots training and validation accuracy and loss curves.
    `history` is the object returned by model.fit().
    """
    plt.figure(figsize=(14, 5))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

plot_training_history(model.history)
```



Testing the Model with The Test Data

In [11]:
```
loss, acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {acc:.2f}")
```

```
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9783 - loss: 0.1050
Test Accuracy: 0.98
```

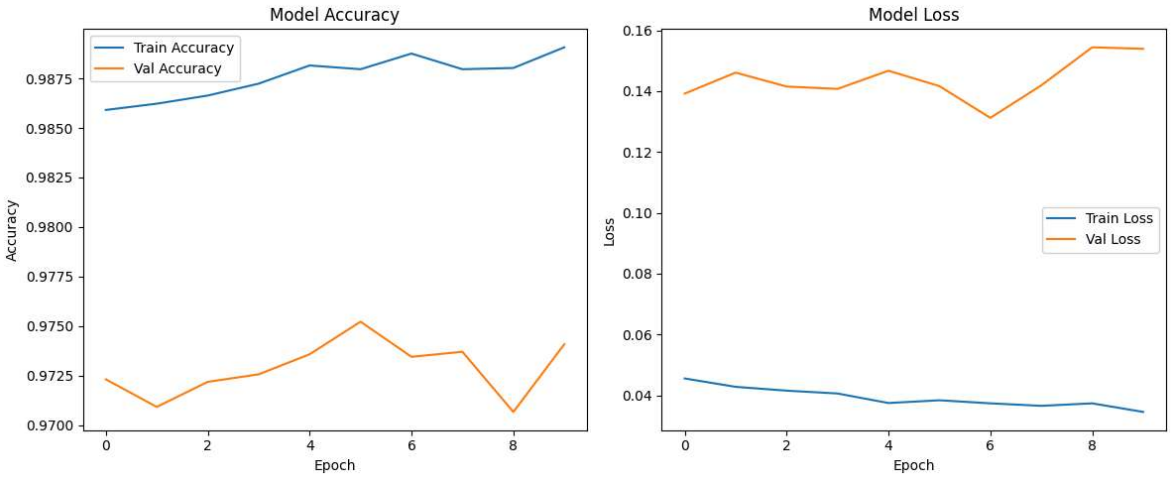
Validating the Model with unseen Data

In [12]:
```
loss, acc = model.evaluate(X_val, y_val)
print(f"Test Accuracy: {acc:.2f}")
```

```
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9723 - loss: 0.1315
Test Accuracy: 0.97
```

In [13]:
```
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split

import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot training & validation loss values
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

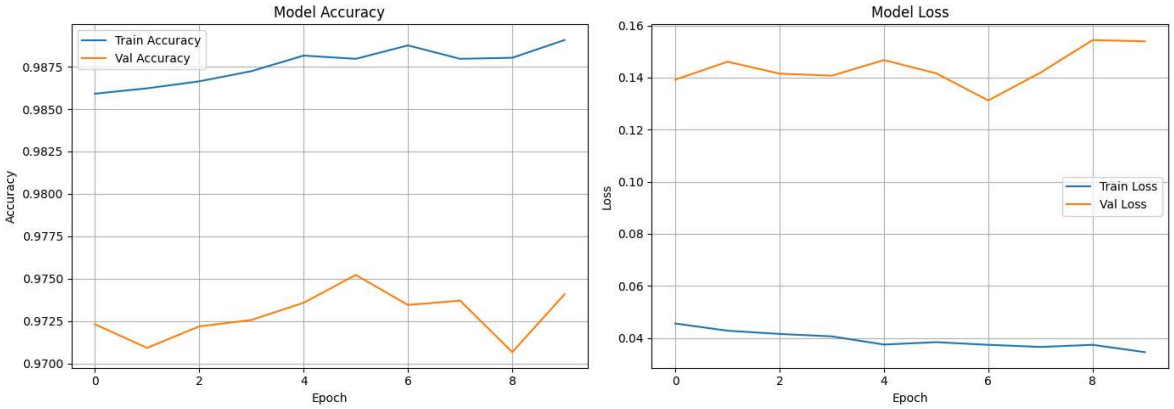
```
Epoch 1/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 10s 10ms/step - accuracy: 0.9852 - loss: 0.0448 - va
l_accuracy: 0.9723 - val_loss: 0.1392
Epoch 2/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9873 - loss: 0.0434 - val_
accuracy: 0.9709 - val_loss: 0.1461
Epoch 3/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9867 - loss: 0.0393 - val_
accuracy: 0.9722 - val_loss: 0.1415
Epoch 4/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.9878 - loss: 0.0379 - val_
accuracy: 0.9726 - val_loss: 0.1407
Epoch 5/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9876 - loss: 0.0377 - val_
accuracy: 0.9736 - val_loss: 0.1467
Epoch 6/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9872 - loss: 0.0384 - val_
accuracy: 0.9752 - val_loss: 0.1417
Epoch 7/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9892 - loss: 0.0337 - val_
accuracy: 0.9735 - val_loss: 0.1312
Epoch 8/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9884 - loss: 0.0344 - val_
accuracy: 0.9737 - val_loss: 0.1419
Epoch 9/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9888 - loss: 0.0332 - val_
accuracy: 0.9707 - val_loss: 0.1544
Epoch 10/10
989/989 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9898 - loss: 0.0324 - val_
accuracy: 0.9741 - val_loss: 0.1539
```



Plot Training and Validation Accuracy/Loss

In [14]:
```python
import matplotlib.pyplot as plt

def plot_training_history(history):
    """
    Plots training and validation accuracy and loss curves.
    `history` is the object returned by model.fit().
    """
    plt.figure(figsize=(14, 5))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
```

```
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()


plot_training_history(history)
```

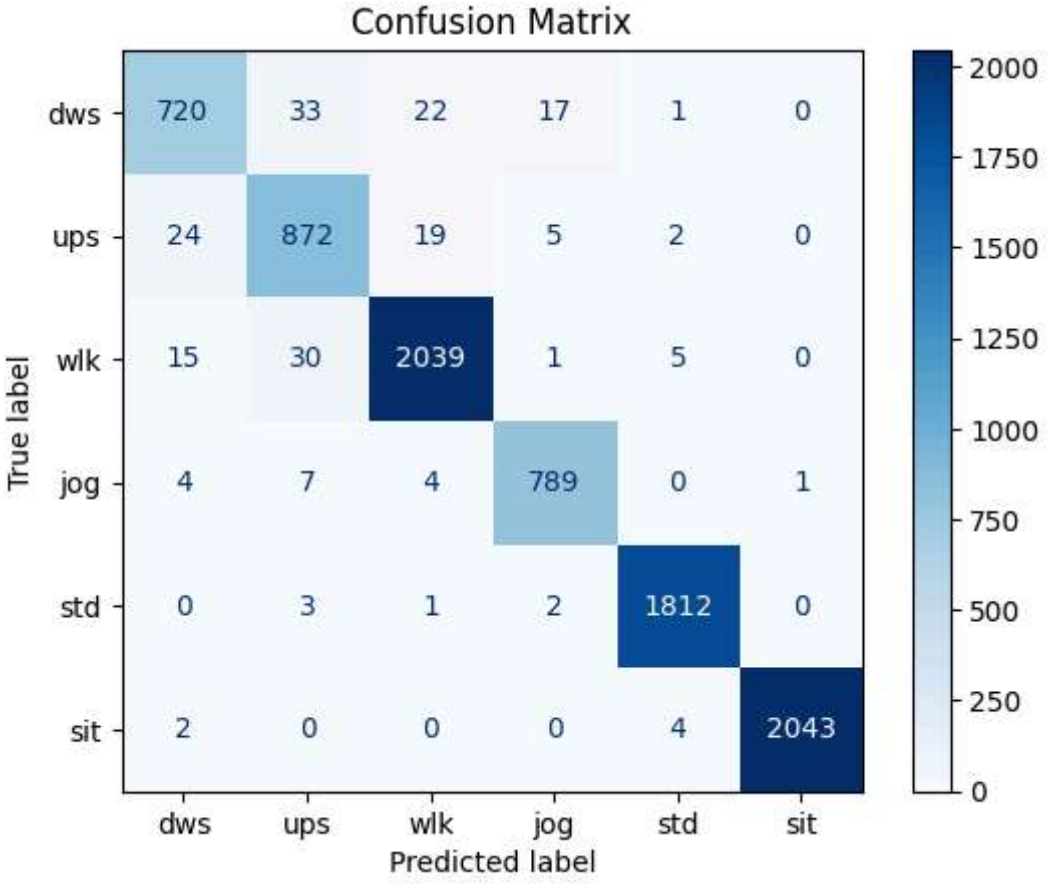


Confusion Matrix (with heatmap)

In [15]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

def plot_confusion_matrix(model, X_test, y_test, class_names, use_softmax=True):
    """
    Plots a confusion matrix for a trained Keras model.
    Assumes y_test is either integer labels (sparse) or one-hot.
    """
    # If labels are one-hot encoded, convert to integer
    if y_test.ndim == 2:
        y_true = np.argmax(y_test, axis=1)
    else:
        y_true = y_test

    # Predict class labels
    y_pred = model.predict(X_test)
    y_pred = np.argmax(y_pred, axis=1) if use_softmax else y_pred

    # Confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_name
```

```
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.grid(False)
    plt.show()
plot_confusion_matrix(model, X_test, y_test, class_names=ACT_LABELS)
```

```
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



Classification Report

In [16]:

```
from sklearn.metrics import classification_report

def print_classification_report(model, X_test, y_test, class_names, use_softmax=
    """
    Prints precision, recall, F1-score for each class.
    """
    # Handle one-hot labels
    if y_test.ndim == 2:
        y_true = np.argmax(y_test, axis=1)
    else:
        y_true = y_test

    # Get predictions
    y_pred = model.predict(X_test)
    y_pred = np.argmax(y_pred, axis=1) if use_softmax else y_pred

    print("Classification Report:")
    print(classification_report(y_true, y_pred, target_names=class_names))
print_classification_report(model, X_test, y_test, class_names=ACT_LABELS)
```

```
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Classification Report:
              precision    recall  f1-score   support

         dws       0.94      0.91      0.92       793
         ups       0.92      0.95      0.93       922
         wlk       0.98      0.98      0.98      2090
         jog       0.97      0.98      0.97       805
         std       0.99      1.00      1.00      1818
         sit       1.00      1.00      1.00      2049

    accuracy                           0.98      8477
   macro avg       0.97      0.97      0.97      8477
weighted avg       0.98      0.98      0.98      8477
```