Early Detection of Parkinson’s Disease Using Deep Learning Models

MSc Research Project

Data Analytics

Abhijeet Nautiyal

Student ID: 22165835

School of Computing

National College of Ireland

Supervisor: Prof. Jaswinder Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abhijeet Nautiyal
Student ID:	22165835
Programme:	MSc in Data Analytics
	Year: 2024-2025.
Module:	Research Project
Supervisor:	Prof. Jaswinder Singh
Submission Due Date:	12-12-2024
Project Title:	Early Detection of Parkinson's Disease Using Deep Learning Models
Word Count: X	**Page Count**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ………………………………………………………………

Date: 12-12-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Early Detection of Parkinson's Disease Using Deep Learning Models (GRU and LSTM)

Abhijeet Nautiyal

x22165835

1. Hardware Requirement

Hardware used for this research is a HP Elitebook with 8 gb ram and windows 10 operating system with Intel i5 processor as shown in Fig 1

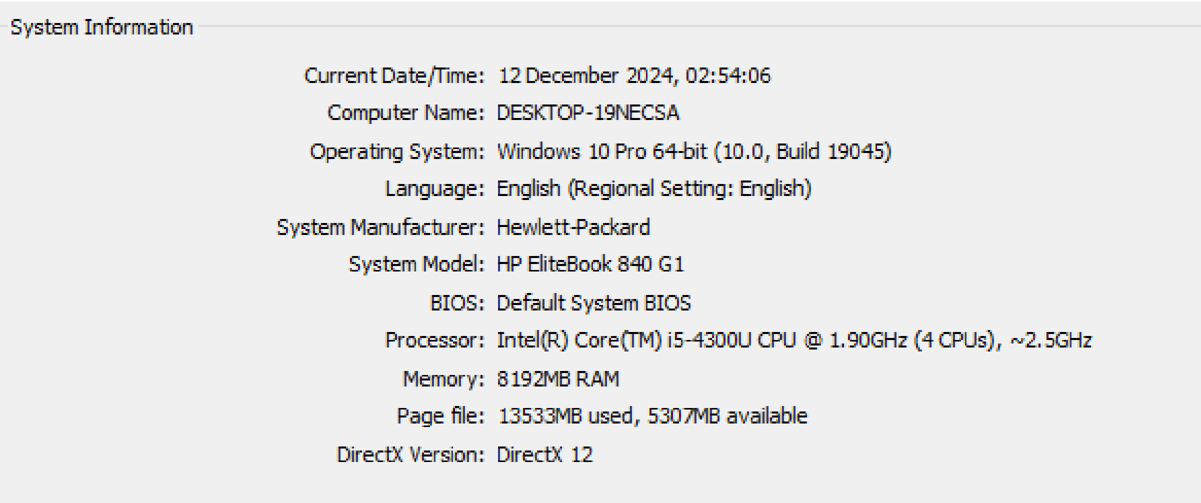

Fig 1: Hardware Requirement

2. Software Requirement

Software used in order to complete the research was Jupyter Notebook under Anaconda navigator, python programming language and Google Colab

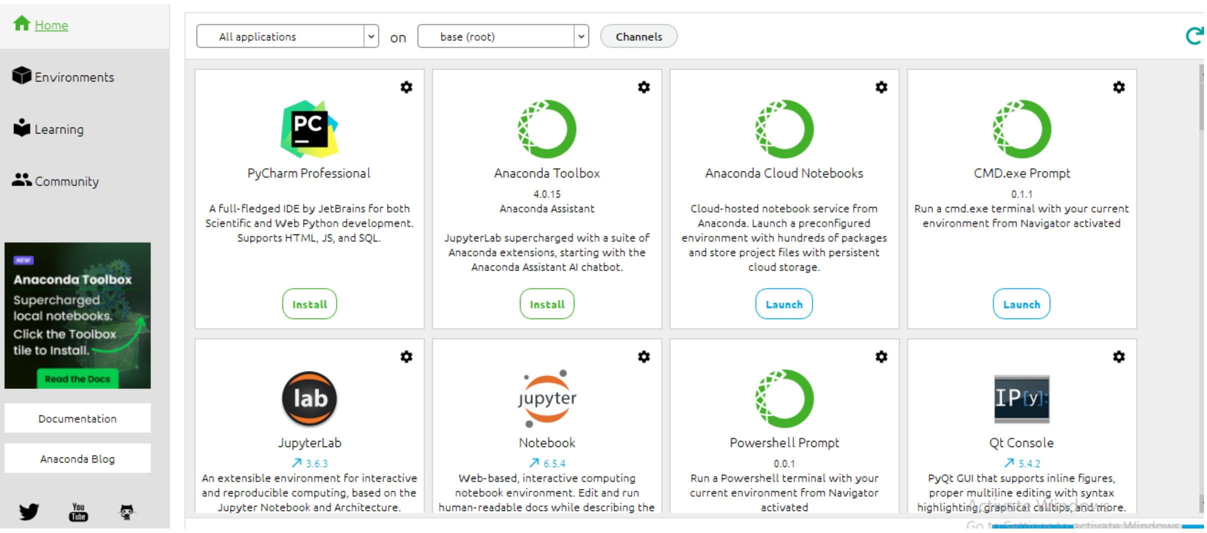

Fig 2 Software Requirement

3. Implementation

Only one python notebook file “Parkinson,ipynb” has been used in entire research project to implement the code

Below is the list of libraries required to implement the pipeline. These libraries facilitate data preprocessing, visualization, feature extraction, and model development.

- **Core Libraries**:
  - `numpy`: For numerical operations and array manipulations.
  - `pandas`: For data manipulation and analysis.
  - `matplotlib` and `seaborn`: For data visualization and exploratory analysis.
- **Data Preprocessing and Feature Engineering**:
  - `sklearn` (scikit-learn): For preprocessing (label encoding, normalization, PCA) and model evaluation (cross-validation, metrics).
- **Deep Learning**:
  - `tensorflow` and `keras`: For building and training the GRU and LSTM models.
- **Audio Processing** :
  - `librosa`: For feature extraction from audio files (e.g., jitter, shimmer, harmonic-to-noise ratio).
- **Optional Libraries**:
  - `audiomentations`: For audio data augmentation.
  - `joblib`: For saving and loading preprocessed datasets or trained models.

4. Dataset Description:

The publicly available UCI Parkinson's Telemonitoring dataset, the dataset have high-quality vocal recordings that are labelled as either PD or healthy. The dataset is available on https://figshare.com/articles/dataset/Voice_Samples_for_Patients_with_Parkinson_s_Disease_and_Healthy_Controls/23849127?file=41836707

It consist 81 patients audio files, healthy and Parkinson disease patients

5. Data Preprocessing

Preprocessing ensures the data is in a format suitable for machine learning models:

1. **Data Cleaning**:
   - Drop irrelevant or NaN-containing columns (e.g., `mdvp_Shimmer`, `mdvp_Shimmer(dB)`).

```
<ipython-input-4-a264ec144d2d>:45: RuntimeWarning: invalid value encountered in divide
  features["mdvp_Shimmer(dB)"] = 20 * np.log10(np.mean(shimmer_diff / amplitude[:-1]) + 1e-10)
   mdvp_fo_hz  mdvp_fhi_hz  mdvp_flo_hz  mdvp_Jitter(%)  mdvp_Jitter(abs)  \
0  132.737517   159.659677   128.937031        0.443856          0.607848
1  110.737726   115.535270   108.422687        0.429902          0.477954
2  234.140832   246.228883   224.492410        0.283131          0.665399
3  102.910798   106.560050   101.161944        0.216765          0.223558
4  111.704933   116.204559   107.798218        0.527576          0.589521

   mdvp_rap  mdvp_ppq  Jitter:DDP  mdvp_Shimmer  mdvp_Shimmer(dB)  ...  \
0  0.786927  2.199056    2.360782    457.126331         13.200725  ...
1  0.784348  2.617849    2.353044           NaN               NaN  ...
2  0.642065  2.136629    1.926194           NaN               NaN  ...
3  0.606771  2.105766    1.820313           NaN               NaN  ...
4  1.030761  3.171679    3.092283    171.860480          4.703520  ...

       rpde       dfa   spread1   spread2        d2       ppe  \
0  4.851480  3.362567  0.000006  0.087086  0.210967  1.736624
1  4.553787  1.661216  0.000026  0.028616  0.099528  1.071116
2  4.812134  3.705580  0.000462  0.081134  0.196838  1.615919
3  4.795740  2.879825  0.000031  0.053941  0.154645  1.508131
4  4.851945  1.467221 -0.001471  0.029912  0.117399  1.229219

                                   Sample ID  Label   Age Sex
0  AH_064F_7AB034C9-72E4-438B-A9B3-AD7FDA1596C5     HC  69.0   M
1  AH_114S_A89F3548-0B61-4770-B800-2E26AB3908B6     HC  43.0   M
2  AH_121A_BD5BA248-E807-4CB9-8B53-47E7FFE5F8E2     HC  18.0   F
3  AH_123G_559F0706-2238-447C-BA39-DB5933BA619D     HC  28.0   M
```

✓ 1s completed at 9:31 PM

Fig3 NAN Column

```python
# Drop the specified columns containing NaN values
columns_to_drop = ['mdvp_Shimmer', 'mdvp_Shimmer(dB)']
df.drop(columns=columns_to_drop, inplace=True)

# Rename the 'Label' column to 'status'
df.rename(columns={'Label': 'status'}, inplace=True)
```

Fig4 NAN Column Drop and Rename Status Column

- Rename the `Label` column to `status` for clarity.

2. **Feature Engineering**:
   - Extract features like jitter, shimmer, and HNR using `librosa` (if working with raw audio).
   - Use extracted feature datasets for machine learning.

```
[ ] features_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 23 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   mdvp_fo_hz         81 non-null     float64
 1   mdvp_fhi_hz        81 non-null     float64
 2   mdvp_flo_hz        81 non-null     float64
 3   mdvp_Jitter(%)     81 non-null     float64
 4   mdvp_Jitter(abs)   81 non-null     float64
 5   mdvp_rap           81 non-null     float64
 6   mdvp_ppq           81 non-null     float64
 7   Jitter:DDP         81 non-null     float64
 8   mdvp_Shimmer       44 non-null     float64
 9   mdvp_Shimmer(dB)   44 non-null     float64
 10  Shimmer:dda        81 non-null     float32
 11  nhr                81 non-null     float32
 12  hnr                81 non-null     float64
 13  rpde               81 non-null     float64
 14  dfa                81 non-null     float32
 15  spread1            81 non-null     float64
 16  spread2            81 non-null     float32
 17  d2                 81 non-null     float64
 18  ppe                81 non-null     float64
 19  Sample ID          81 non-null     object
 20  Label              81 non-null     object
 21  Age                81 non-null     float64
 22  Sex                81 non-null     object
dtypes: float32(4), float64(16), object(3)
memory usage: 13.4+ KB
```

Fig5 Feature Extracted

3. **Normalization**:
   - Apply normalization using `StandardScaler` to scale features to a mean of 0 and unit variance.
4. **Dimensionality Reduction**:
   - Use PCA to reduce dimensionality while retaining significant variance.
5. **Data Augmentation**:
   - Apply pitch shifting, time-stretching, or noise injection for audio. For tabular data, simulate variations by adding random noise to feature columns.

```python
# Data Augmentation - Add noise to features
def augment_data(df, noise_factor=0.05):
    augmented_df = df.copy()
    for column in df.columns:
        # Check if the column contains numeric data before applying noise
        if column != 'status' and pd.api.types.is_numeric_dtype(df[column]):
            noise = np.random.normal(0, noise_factor, df[column].shape)
            augmented_df[column] = df[column] + noise
    return augmented_df

augmented_df = augment_data(df)

# Principal Component Analysis (PCA)
pca = PCA(n_components=10)  # Choose the number of components based on your analysis
pca_features = pca.fit_transform(df.drop(columns=['status']).select_dtypes(include=['number']))

# Create a DataFrame with the transformed features
pca_df = pd.DataFrame(pca_features, columns=[f'PC{i+1}' for i in range(pca.n_components_)])
pca_df['status'] = df['status'].values  # Add the label back to the DataFrame

# Normalization - Standard Scaling
scaler = StandardScaler()

# Select only numerical features for scaling
numerical_features = df.drop(columns=['status']).select_dtypes(include=['number'])

# Normalize the numerical features
normalized_features = scaler.fit_transform(numerical_features)
```

Fig.6 PCA and Augmentation

6. **Data Splitting**:
    - Split the dataset into training (80%), validation (10%), and testing (10%) subsets.

6. Model Implementation

The deep learning models used in this study are **GRU** and **LSTM**, which are tailored to handle sequential data like voice recordings.

Training Configuration:

- Optimizer: Adam optimizer for adaptive learning rates.
- Loss Function: Binary cross-entropy for binary classification tasks.
- Metrics: Accuracy, precision, recall, and F1 score.
- Epochs: 20-50 (or until convergence).
- Batch Size: 32.

Implementation:

- Use TensorFlow/Keras to define and compile the model.
- Train the model with early stopping or cross-validation to avoid overfitting.

Experiments:

There's 3 model variable with different configuration Fig 7 that has been used to conduct different experiments for LSTM and GRU, this has been applied to see the changes in accuracy and how the model perform with different configuration

```
    return model

# Experiment 1: Default values
model_1 = build_model()

# Experiment 2: Modified parameters
model_2 = build_model(conv_filters=64, kernel_size=1, pool_size=1, gru_units_1=75, gru_units_2=100, dropout_rate=0.3)


#Experiment 3:
model_3 = build_model(conv_filters=64, kernel_size=1, pool_size=1, gru_units_1=100, gru_units_2=125, dropout_rate=0.5)

# Train and evaluate model
early_stopping = EarlyStopping(monitor='val_loss', patience=10, mode='min', verbose=1)
history = model_1.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), batch_size=32, epochs=40, callbacks =[early_stopping])
# history = model_2.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), batch_size=32, epochs=40)
# history = model_3.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), batch_size=32, epochs=40)

# Extract metrics from training history
```

```
# Experiment 1: Default values
model_1 = build_model()

# Experiment 2: Modified parameters
model_2 = build_model(conv_filters=64, kernel_size=1, pool_size=1, lstm_units_1=75, lstm_units_2=100, dropout_rate=0.3)

#Experiment 3:
model_3 = build_model(conv_filters=64, kernel_size=1, pool_size=1, lstm_units_1=100, lstm_units_2=125, dropout_rate=0.5)



# Train and evaluate model
early_stopping = EarlyStopping(monitor='val_loss', patience=10, mode='min', verbose=1)
history = model_1.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), batch_size=32, epochs=40, callbacks =[early_stopping])
# history = model_2.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), batch_size=32, epochs=40)
# history = model_3.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), batch_size=32, epochs=40)
```

Fig 7 Model experiments GRU and LSTM

After enabling the model_1 as you can see in the Fig 7, you'd have to make the changes in Train and evaluate part of the code as well and then changes in predictions variable and also in Evaluate_Performance function Fig 8

```
# Predict and evaluate
predictions = (model_1.predict(X_test_scaled) > 0.5).astype("int32")

def Evaluate_Performance(Model, Xtrain, Xtest, Ytrain, Ytest, Predictions):
    print("\n • Training Accuracy Score : ", round(Model.evaluate(Xtrain, Ytrain, verbose=0)[1] * 100, 2))
    print("\n • Testing Accuracy Score : ", round(Model.evaluate(Xtest, Ytest, verbose=0)[1] * 100, 2))
    print(' • Precision Score is :', round(precision_score(Ytest, Predictions) * 100, 2))
    print(' • Recall Score is :', round(recall_score(Ytest, Predictions) * 100, 2))
    print(' • F1-Score Score is :', round(f1_score(Ytest, Predictions) * 100, 2))
    print('-' * 80)

    conf_matrix = confusion_matrix(Ytest, Predictions)
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap=plt.cm.Blues, annot_kws={"size": 16})
    plt.title('Predicted Labels', y=1.05, fontsize=20, fontfamily='Times New Roman')
    plt.ylabel('True Labels', labelpad=25, fontsize=20, fontfamily='Times New Roman')
    plt.show()
    print('-' * 80)

Evaluate_Performance(model_1, X_train_scaled, X_test_scaled, y_train, y_test, predictions)
```

Fig 8 Predict and Evaluate Change

Fig 8 and Fig 9 is the summary of evalution of Model_1, similarly you can change the variable to Model_2 and Model_3 to see the how the other configuration works on LSTM and GRU

```
Evaluate_Performance(model_1, X_train_scaled, X_test_scaled, y_train, y_test, predictions)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 9s 778ms/step - accuracy: 0.5208 - loss: 0.6934 - val_accuracy: 0.5294 - val_loss: 0.6933
Epoch 2/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - accuracy: 0.5625 - loss: 0.6930 - val_accuracy: 0.4706 - val_loss: 0.6934
Epoch 3/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.5729 - loss: 0.6924 - val_accuracy: 0.6471 - val_loss: 0.6936
Epoch 4/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.5521 - loss: 0.6921 - val_accuracy: 0.4706 - val_loss: 0.6938
Epoch 5/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - accuracy: 0.5521 - loss: 0.6918 - val_accuracy: 0.4118 - val_loss: 0.6941
Epoch 6/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - accuracy: 0.5625 - loss: 0.6910 - val_accuracy: 0.4118 - val_loss: 0.6943
Epoch 7/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step - accuracy: 0.5104 - loss: 0.6913 - val_accuracy: 0.4118 - val_loss: 0.6946
Epoch 8/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - accuracy: 0.5417 - loss: 0.6902 - val_accuracy: 0.4118 - val_loss: 0.6948
Epoch 9/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - accuracy: 0.5521 - loss: 0.6889 - val_accuracy: 0.4118 - val_loss: 0.6951
Epoch 10/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step - accuracy: 0.5833 - loss: 0.6873 - val_accuracy: 0.4118 - val_loss: 0.6956
Epoch 11/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.5104 - loss: 0.6872 - val_accuracy: 0.4118 - val_loss: 0.6960
```

Fig 8 CNN + LSTM Model Evaluation

```
Evaluate_Performance(model_1, X_train_scaled, X_test_scaled, y_train, y_test, predictions)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 9s 512ms/step - accuracy: 0.4792 - loss: 0.6925 - val_accuracy: 0.4118 - val_loss: 0.6938
Epoch 2/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.5833 - loss: 0.6901 - val_accuracy: 0.4118 - val_loss: 0.6951
Epoch 3/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5208 - loss: 0.6901 - val_accuracy: 0.4118 - val_loss: 0.6963
Epoch 4/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5521 - loss: 0.6888 - val_accuracy: 0.4118 - val_loss: 0.6976
Epoch 5/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4896 - loss: 0.6881 - val_accuracy: 0.4118 - val_loss: 0.6979
Epoch 6/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.5729 - loss: 0.6812 - val_accuracy: 0.4118 - val_loss: 0.6989
Epoch 7/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.5208 - loss: 0.6838 - val_accuracy: 0.4118 - val_loss: 0.6991
Epoch 8/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5729 - loss: 0.6796 - val_accuracy: 0.4118 - val_loss: 0.6997
Epoch 9/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.5938 - loss: 0.6758 - val_accuracy: 0.4118 - val_loss: 0.7002
Epoch 10/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.5833 - loss: 0.6740 - val_accuracy: 0.4118 - val_loss: 0.7005
Epoch 11/40
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.6458 - loss: 0.6648 - val_accuracy: 0.4118 - val_loss: 0.7013
```

Fig 9 CNN + GRU Model Evaluation