Diagnosing-ADHD-With-ConvLSTM-master

Abstract

Attention Deficit Hyperactivity Disorder (ADHD) is a neurodevelopmental disorder that affects both children and adults, significantly impacting daily functioning and quality of life. Early and accurate detection of ADHD is crucial for timely intervention and effective management of the condition. In recent years, deep learning models have demonstrated remarkable success in various medical applications, prompting exploration of their potential in ADHD diagnosis.

This research paper proposes a novel Convolutional Long Short-Term Memory (conv-LSTM) model for detecting ADHD, leveraging the strengths of both convolutional neural networks (CNNs) and long short-term memory networks (LSTMs). The conv-LSTM architecture integrates spatial and temporal dependencies within neuroimaging data, providing a robust framework for capturing intricate patterns indicative of ADHD.

We conducted experiments on a comprehensive dataset comprising neuroimaging scans from individuals with diagnosed ADHD and healthy controls. The proposed conv-LSTM model demonstrated superior performance compared to traditional machine learning approaches and other deep learning architectures. Our model not only achieved high accuracy but also exhibited improved interpretability, contributing to its potential clinical utility.

Furthermore, interpretability analyses were conducted to enhance the understanding of the features learned by the conv-LSTM model, shedding light on the neuroanatomical regions and temporal dynamics associated with ADHD pathology. The results suggest that the proposed model can provide valuable insights into the underlying neurobiological mechanisms of ADHD.

In conclusion, this research contributes to the growing body of literature on the application of deep learning in medical diagnostics, specifically focusing on ADHD detection. The conv-LSTM model shows promise as an effective and interpretable tool for early ADHD identification, opening avenues for future research in leveraging advanced machine learning techniques for improved understanding and management of neurodevelopmental disorders.

Title: Convolutional Long Short-Term Memory Networks for the Detection of ADHD: Unveiling Patterns in Neuroimaging Data

Introduction:

Attention Deficit Hyperactivity Disorder (ADHD) is a prevalent neurodevelopmental disorder characterized by persistent patterns of inattention, hyperactivity, and impulsivity. The accurate and timely diagnosis of ADHD is crucial for effective intervention and management of the condition. In recent years, advancements in neuroimaging techniques have provided researchers with unprecedented access to structural and functional information about the brain, opening new avenues for developing sophisticated models capable of discerning subtle patterns indicative of ADHD

This research paper introduces a novel approach to ADHD detection utilizing Convolutional Long Short-Term Memory (conv-LSTM) networks, a deep learning architecture that combines the strengths of both convolutional neural networks (CNNs) and long short-term memory networks (LSTMs). The conv-LSTM model is designed to leverage the spatial hierarchies captured by CNNs and the temporal dependencies modeled by LSTMs, offering a comprehensive framework for extracting intricate patterns from neuroimaging data.

The conventional methods for ADHD diagnosis often rely on clinical evaluations, behavioral assessments, and self-reported information. While informative, these methods may lack the granularity required to unveil subtle neuroanatomical and functional nuances associated with ADHD. In contrast, neuroimaging data, such as structural magnetic resonance imaging (sMRI) and functional magnetic resonance imaging (fMRI), provide rich and multidimensional information about the brain's architecture and activity. By harnessing the power of conv-LSTM models, we aim to enhance the accuracy and sensitivity of ADHD detection by discerning intricate patterns within these complex neuroimaging datasets.

In this paper, we delve into the theoretical underpinnings of the conv-LSTM model and discuss its potential in capturing both spatial and temporal dependencies inherent in neuroimaging data. We present a comprehensive review of existing methodologies for ADHD detection, highlighting the limitations of traditional approaches and emphasizing the need for advanced computational models. Furthermore, we describe the dataset used in this study, consisting of carefully curated neuroimaging data from individuals with and without ADHD, to evaluate the efficacy of the proposed conv-LSTM model.

By exploring the intersection of deep learning and neuroimaging, this research seeks to contribute to the growing body of knowledge aimed at improving the precision and reliability of ADHD diagnosis. The subsequent sections of this paper will elaborate on the methodology, experimental setup, results, and discussion, providing insights into the potential of conv-LSTM models in advancing our understanding of ADHD through the analysis of neuroimaging data.

Methods Proposed:

1. Data Collection:
   - Gather a dataset comprising brain imaging data or relevant biomarkers associated with ADHD.
   - Include a control group without ADHD for comparison.

2. Data Preprocessing:
   - Normalize and standardize the input data to ensure consistency.
   - Handle missing data appropriately.

- Augment the dataset if necessary to prevent overfitting.

3. Feature Extraction:
   - If dealing with brain imaging data (e.g., fMRI, EEG), extract relevant features using techniques like spatial filtering, temporal filtering, or other domain-specific methods.

4. Model Architecture Design:
   - Design a Convolutional Long Short-Term Memory (conv-LSTM) model architecture. A conv-LSTM combines the spatial filters of CNNs with the sequential modeling capabilities of LSTMs.
   - Specify the number of layers, filter sizes, and other hyperparameters.
   - Consider including dropout layers for regularization.

5. Training:
   - Split the dataset into training, validation, and test sets.
   - Train the conv-LSTM model on the training set using an appropriate optimization algorithm (e.g., Adam, RMSprop).
   - Monitor the model's performance on the validation set to avoid overfitting.

6. Hyperparameter Tuning:
   - Fine-tune hyperparameters such as learning rate, batch size, and others based on validation performance.

7. Evaluation:
   - Evaluate the trained model on the test set to assess its generalization performance.
   - Use relevant metrics such as accuracy, precision, recall, F1-score, or area under the receiver operating characteristic (ROC) curve.

8. Comparison with Baselines:
   - Compare the performance of the conv-LSTM model with existing models or baseline methods to demonstrate its effectiveness.

9. Interpretability and Visualization:
   - Provide insights into which parts of the input data contribute to the model's decisions.
   - Visualize activation maps or relevant features to enhance interpretability.

10. Discussion and Conclusion:
    - Discuss the results, limitations, and potential areas for improvement.
    - Conclude with the implications of the findings for ADHD detection and potential future work.

FMRIDataGenerator

The code defines a custom data generator class `FMRIDataGenerator` for handling MRI (Magnetic Resonance Imaging) data in the context of a Keras-based deep learning model. This generator is designed to be used with the Keras `fit_generator` function to feed data into a neural network during training.

Here's a breakdown of the code workings:

1. Importing Libraries:
   - `numpy` (`np`): Used for numerical operations.
   - `tensorflow.keras` (`keras`): The deep learning library for building and training neural networks.
   - `scipy.ndimage.zoom`: Used for zooming (resizing) 3D images.
   - `os`: Provides a way of interacting with the operating system.
   - `nibabel` (`nib`): Used for reading and writing neuroimaging data in the NIfTI format.

2. Class Definition (`FMRIDataGenerator`):
   - Inherits from `keras.utils.Sequence`, which is a base class for Keras data generators.

3. Initialization Method (`__init__`):
   - Initializes various parameters needed for the data generator.
   - `list_IDs`: List of image IDs (file names).
   - `labels`: Dictionary mapping image IDs to their corresponding labels.
   - `dataset_dir`: Directory where the MRI data is stored.
   - `batch_size`: Number of samples per batch.

4. Length Method (`__len__`):
   - Returns the number of batches per epoch.

5. Get Item Method (`__getitem__`):
   - Generates one batch of data.
   - Retrieves the indexes of the current batch.
   - Finds the corresponding image IDs for the batch.
   - Calls `__data_generation` to generate the actual data (features and labels).

6. On Epoch End Method (`on_epoch_end`):
   - Updates indexes after each epoch.
   - Shuffles the indexes if the `shuffle` parameter is set to `True`.

7. Data Generation Method (`__data_generation`):

   - Generates data containing `batch_size` samples.
   - Calls `preprocess_image` for each image in the batch.
   - Returns a tuple `(X, y)` where `X` is the input data (MRI images) and `y` is the corresponding label.

8. Preprocess Image Method (`preprocess_image`):
   - Loads an MRI image using `nibabel`.
   - Depending on the number of time steps (`self.time_length`), it either truncates, pads, or zooms the image.
   - Zooming involves resizing the image using `scipy.ndimage.zoom`.
   - Returns the preprocessed 4D MRI image.

9. Truncate Image Method (`truncate_image`):
   - Truncates the image to the specified time length.

10. Pad Image Method (`pad_image`):
    - Pads the image with zeros to match the specified time length.

This generator is designed for use with 4D MRI data where each volume has dimensions (time, x, y, z). It handles variations in the number of time steps by truncating, padding, or zooming the images accordingly. The preprocessed data is then used as input for training a neural network.

ADHD FMRI Classification

The code for training a deep learning model, specifically a convolutional neural network (CNN) with a long short-term memory (LSTM) layer, on functional magnetic resonance imaging (fMRI) data.

- Warnings and Logging

```
import warnings
warnings.filterwarnings("ignore")

import logging
```

These lines suppress warning messages and set the logging level to FATAL for TensorFlow, meaning only fatal error messages will be shown.

- Importing Libraries

```
from Code.data_generator import FMRIDataGenerator
import numpy as np
import pandas as pd
import os
import sys
from datetime import datetime
import tensorflow as tf
from tensorflow.keras.layers import Conv3D, MaxPool3D, TimeDistributed, Flatten, LSTM,
Dense
from tensorflow.keras import Sequential
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import CSVLogger
import tensorflow.keras as keras
```

The necessary libraries and modules are imported, including TensorFlow, NumPy, pandas, and others.

- Data Work

```
file_num = sys.argv[1]

# Dataframes
dataset_dir = "/pylon5/cc5614p/deopha32/fmri_images/model_data/"
model_train_data = pd.read_csv("/home/deopha32/ADHD-
FMRI/Data/training_data_{}".format(file_num))
model_val_data = pd.read_csv("/home/deopha32/ADHD-
FMRI/Data/validatation_data_{}".format(file_num))
```

The script takes a command-line argument `file_num` and reads training and validation data from CSV files.

```
partition = {'train': model_train_data['Image'].values, 'validation':
model_val_data['Image'].values}
```

Creates a partition dictionary for training and validation data.

```
train_labels = {}
for index, row in model_train_data.iterrows():
    train_labels[row['Image']] = row['DX']
```

Creates a dictionary `train_labels` with image names as keys and diagnosis labels as values for the training data. Similar steps are done for validation data.

- Model Meta

```
epochs = 500
batch_size = 6
input_shape = (177, 28, 28, 28, 1)
train_steps_per_epoch = model_train_data.shape[0] // batch_size
validate_steps_per_epoch = model_val_data.shape[0] // batch_size
```

Defines training parameters like epochs, batch size, and input shape.

- Data Generators

```
training_generator = FMRIDataGenerator(partition['train'], train_labels, dataset_dir,
batch_size)
validation_generator = FMRIDataGenerator(partition['validation'], val_labels, dataset_dir,
batch_size)
```

Creates data generators using the `FMRIDataGenerator` class.

```
curr_time = f'{datetime.now():%H-%M-%S%z_%m%d%Y}'
logger_path = "/pylon5/cc5614p/deopha32/Saved_Models/adhd-fmri-
history_cv{num}_{time}.csv".format(num=file_num, time=curr_time)
```

Generates a timestamped path for saving the training history in a CSV file.

```
csv_logger = CSVLogger(logger_path, append=True)
```

```
callbacks = [csv_logger]
```

Creates a CSV logger callback to log training history.

- Model Architecture

```
with tf.device('/gpu:0'):
   cnn_lstm_model = Sequential()
   # ... (layers for GPU)
```

Defines a CNN-LSTM model architecture, specifically using GPU for layers that support parallel processing.

```
with tf.device('/cpu:0'):
   # ... (layers for CPU)
```

Defines LSTM layer using CPU for sequential processing.

```
with tf.device('/gpu:0'):
   # ... (layers for GPU)
```

Defines additional layers using GPU.

```
cnn_lstm_model.compile(optimizer=optimizers.Adam(lr=0.0001),
           loss='binary_crossentropy',
           metrics=['accuracy'])
```

Compiles the model with Adam optimizer, binary crossentropy loss, and accuracy metric.

```
cnn_lstm_model.fit_generator(generator=training_generator,
   steps_per_epoch=train_steps_per_epoch, verbose=1, callbacks=callbacks,
   validation_data=validation_generator, validation_steps=validate_steps_per_epoch,
   epochs=epochs)
```

Fits the model using the defined data generators, steps, callbacks, and training parameters.

Overall, this is a pipeline for training a deep learning model on fMRI data using a combination of CNN and LSTM layers. The model is trained with GPU acceleration, and training progress is logged to a CSV file.