CS577-Deep Learning: Project Final Report
Child Sleep State Detection from Wrist-Worn Accelerometer Data Using A Faster R-CNN Model

Xue Zhang
Illinois Institute of Technology
Chicago, Illinois, USA
xzhang143@hawk.iit.edu
A20494478

Hongbo Wang
Illinois Institute of Technology
Chicago, Illinois, USA
hwang218@hawk.iit.edu
A20524770

ABSTRACT

Sleep state detection is a crucial step when extrapolating patterns from actigraphy data. Numerous supervised detection algorithms have been developed with parameters estimated from and optimized for a particular dataset; however, their generalizability is unknown. Additionally, existing sleep detection methods have several disadvantages, such as relying on feature extraction and failing to recognize temporal sequence patterns in long-term associated data. In this project, our team proposes and validates the Fast R-CNN to detect sleep states from wrist-worn accelerometer data. We first convert all continuous accelerometer data into different images based on their onset and wake-up activities. Then, we use Fast R-CNN for detection. We define the loss function and evaluation metrics based on the competition requirements. Finally, we evaluate the test results based on the best model parameters.

KEYWORDS

Sleep State Detection, Faster R-CNN

ACM Reference Format:
Xue Zhang and Hongbo Wang. 2018. CS577-Deep Learning: Project Final Report Child Sleep State Detection from Wrist-Worn Accelerometer Data Using A Faster R-CNN Model. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Precision in detecting sleep-wake (SW) cycles is crucial for extrapolating sleep patterns from actigraphy data[3]. Actigraphy monitors individual activity levels at 30-second or 60-second intervals consistently over an extended duration, typically spanning from a few days to several weeks. Through analysis of these records, researchers can approximate circadian cycles and SW cycles, as well as derive essential sleep metrics like sleep/wake onset times (S/WOTs), sleep durations, and the within-subject variability of these sleep metrics.

Given the need for cost-effective measurement tools, wearable accelerometers are widely utilized due to their affordability and reasonable accuracy in estimating movement. Numerous algorithms have been developed for estimating SW cycles from wrist-worn accelerometer data, especially in children. Most of these techniques are binary classification algorithms such as logistic regression, linear discriminant classifiers, support vector machines and random forests. Walch[9] derived physical activity features from the accelerometer data and applied k-nearest neighbors (KNN), random forest (RF), and multi-layer perceptron (MLP) separately for sleep state classification. Sundararajan [8] explored random forest-based non-wear detection to gain insight into the potential for daytime nap detection. Sazonov [6] utilized logistic regression and neural networks as predictors to detect sleep-wake states for infants. Domingues [2] proposed a method that combines two linear discriminant classifiers, trained with two different criteria involving movement detection, to generate a first-state estimate. This result is then refined by a Hidden Markov Model-based algorithm to solve the unbalanced state distribution issue.

However, attempts to classify sleep stages from accelerometer-only data either detect sleep with or without a sleep diary from wrist-worn accelerometers by using wrist-acceleration specific features. Actigraphy data collected by different wearable activity trackers for the same types of activity have different ranges and distributions. The parameters of these algorithms are fine-tuned to maximize the detection accuracy for a particular dataset, and their generalizability to other datasets is questionable.

To address generalization issues, some researchers have devised algorithms utilizing matured deep learning models to extract more essential features from accelerometer data. Santos[5] designed a 6-layer CNN for automatic feature extraction. Xia[11] proposed a deep neural network that combines CNN layers with LSTM (LSTM-CNN). Wang[10] used a ResNet-based model as a strong baseline for time series classification, while Song [7] developed a self-attention-based deep learning model. These models achieved decent performance for 2-class (Wake/Sleep) or 3-class (Normal/NREM/REM) sleep stage classification. However, achieving satisfactory performance becomes challenging, especially when dealing with 5 classes, such as Wake, N1, N2, N3, and REM.

In this course project, we explored one of the popular object detection model Faster R-CNN to enhance sleep state classification through a data-driven approach. This model is highly efficient for processing time series data and can significantly improve model performance, extending application fields and overcoming limitations related to the lack of labeled data and noisy datasets. We utilized

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

data from a Kaggle competition and evaluated our approach through cross-validation. Subsequently, we reported the performance of our trained models on previously unseen test data.

In contrast to other published papers, our project's goal is not to produce a classification algorithm that outperforms all existing ones. Instead, we aim to explore the underlying mechanisms of children's sleeping states and investigate how deep learning models can be applied in this field.

2 METHODOLOGY

2.1 Data Collection

The data sets were provided by Global Center for Child & Adolescent Mental Health and were taken from Kaggle to perform the analysis. The data sets comprises about 500 multi-day recordings of wrist-worn accelerometer data annotated with two event types: onset, the beginning of sleep, and wakeup, the end of sleep. Our task is to detect the occurrence of these two events in the accelerometer series.

The descriptions of the data and its columns/features in the dataset are mentioned below. Each data series represents the continuous (multi-day/event) recording for a unique experimental subject.

- `train_series.parquet`: Series to be used as training data. Each series is a continuous recording of accelerometer data for a single subject spanning many days.
  - `series_id`: Unique identifier for each accelerometer series.
  - `step`: An integer time step for each observation within a series.
  - `timestep`: A corresponding datetime with ISO 8601 format
  - `anglez`: The angle of the arm relative to the vertical axis of the body
  - `enmo`: ENMO is the Euclidean Norm Minus One of all accelerometer signals, with negative values rounded to zero. While no standard measure of acceleration exists in this space, this is one of the several commonly computed features
- `test_series.parquet`: Series to be used as the test data, containing the same fields as above. You will predict event occurrences for series in this file.
- `train_events.csv`: Sleep logs for series in the training set recording onset and wake events.
  - `series_id`: Unique identifier for each series of accelerometer data
  - `night`: An enumeration of potential onset / wakeup event pairs. At most one pair of events can occur for each night.
  - `event`: The type of event, whether onset or wakeup.
  - `step and timestamp`: The recorded time of occurrence of the event in the accelerometer series.
- `sample_submission.csv`: A sample submission file in the correct format.

2.2 Data Preprocessing

Sleep logbooks remain the gold standard when working with accelerometer data to detect child sleep states. However, there are many cases where annotation information is not recorded correctly in the given dataset. We have 277 series, but for 8 of these series, there are absolutely no annotations of event data. To explore the

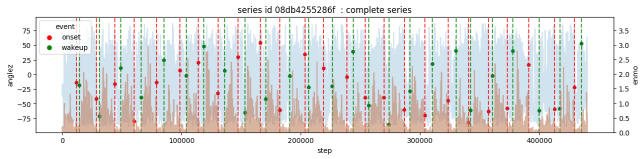

Figure 1: A complete series

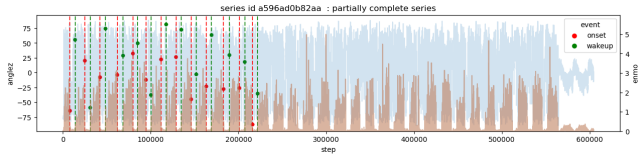

Figure 2: A partially complete series

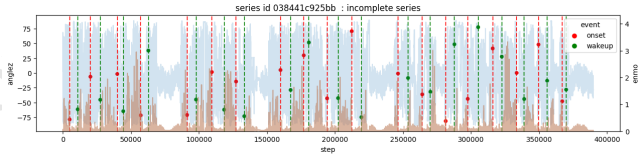

Figure 3: A incomplete series

relationship between accelerometer data and sleep event logs, we divide all training events into complete and incomplete parts based on whether the entry exists or contains null values. Figures 1, 2, and 3 depict a typical complete series, a partially complete series, and an incomplete series, respectively. Given that our task involves detecting the occurrence of two events in the accelerometer series, we need to predict the steps at onset and the steps at wakeup. This involves solving a range problem. We then illustrate the process of converting continuous data into images, which can be utilized by the Faster R-CNN model later and related features that we extracted in the preprocessing process. Essentially, we are interested only in the signal that occurs between 8:30 pm and 8:30 am. Therefore, we define a window span of 24 hours. Each day, we save only the signal within this time period as the image. The descriptions of the updated data and its columns/features in the dataset are mentioned in Table 1 and Table 2.

2.3 Splitting of training, Validation

After preprocessing, we have 7696 entries with 5 attributes in the training data, along with 9585 entries with 10 attributes in the training event annotation data. We split the training dataset into training and validation data (80% for training, 20% for validation).

In this project, we use PyTorch as the development tool. To make the training process efficient, we need to convert all data formats to tensors and use Data Loader to fetch data with a smaller volume than the total amount of data.

Table 1: Features extracted from train event data

series_id	image_name	index_in_series	steps_window	steps_cumulative
038441c925bb	038441c925bb_2018-08-14T20_30_00.jpg	0	720.0	720.0
038441c925bb	038441c925bb_2018-08-15T20_30_00.jpg	1	720.0	720.0
038441c925bb	038441c925bb_2018-08-16T20_30_00.jpg	2	720.0	720.0
038441c925bb	038441c925bb_2018-08-17T20_30_00.jpg	3	720.0	720.0
038441c925bb	038441c925bb_2018-08-18T20_30_00.jpg	4	720.0	720.0
038441c925bb	038441c925bb_2018-08-19T20_30_00.jpg	5	720.0	720.0
038441c925bb	038441c925bb_2018-08-20T20_30_00.jpg	6	17280.0	104400
038441c925bb	038441c925bb_2018-08-21T20_30_00.jpg	7	17280.0	121680.0
038441c925bb	038441c925bb_2018-08-22T20_30_00.jpg	8	17280.0	138960.0
038441c925bb	038441c925bb_2018-08-23T20_30_00.jpg	9	17280.0	156240.0
038441c925bb	038441c925bb_2018-08-24T20_30_00.jpg	10	17280.0	173520.0

Table 2: Features extracted from train event data

series_id	image_name	label	step_in_series	steps_in_window	x0	y0	x1	y1
038441c925bb	038441c925bb_2018-08-15T20_30_00.jpg	onset	4992.0	4272.0.0	336	20	376	380
038441c925bb	038441c925bb_2018-08-15T20_30_00.jpg	wakeup	10932.0	10212.0.0	831	20	871	380
038441c925bb	038441c925bb_2018-08-16T20_30_00.jpg	onset	20244.0	2244.0.0	167	20	207	380
038441c925bb	038441c925bb_2018-08-16T20_30_00.jpg	wakeup	27492	9492.0	771.0	20	811	380
038441c925bb	038441c925bb_2018-08-17T20_30_00.jpg	onset	39996.0	4716.0	373	20	413	380
038441c925bb	038441c925bb_2018-08-17T20_30_00.jpg	wakeup	44400.0	9120.0	740	20	780	380

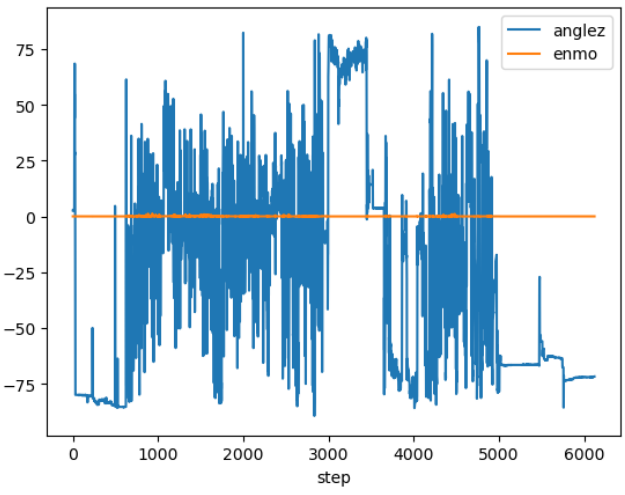

Figure 4: Original continuous data from the accelerometer for one day

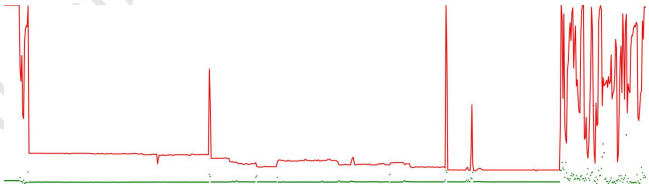

Figure 5: Update image data from the accelerometer for one day

```
In [9]: train_events = pd.read_csv(raw_data_folder/"train_events.csv")
        series_ids = train_events['series_id'].unique()
        num_val_series_ids = round(0.2 * len(series_ids))
        series_ids_in_val = np.random.choice(series_ids, size=num_val_series_ids, replace=False)
        series_ids_in_train = series_ids[~ np.isin(series_ids, series_ids_in_val)]

In [10]: train = window_properties_df.loc[window_properties_df['series_id'].isin(series_ids_in_train)].reset_index(drop=True)
         val = window_properties_df.loc[window_properties_df['series_id'].isin(series_ids_in_val)].reset_index(drop=True)

In [11]: train

Out[11]:
```

	series_id	image_name	idx_in_series	num_steps_window	num_steps_cumulative
0	038441c925bb	038441c925bb_2018-08-14T20_30_00.jpg	0	720.0	720.0
1	038441c925bb	038441c925bb_2018-08-15T20_30_00.jpg	1	17280.0	18000.0
2	038441c925bb	038441c925bb_2018-08-16T20_30_00.jpg	2	17280.0	35280.0
3	038441c925bb	038441c925bb_2018-08-17T20_30_00.jpg	3	17280.0	52560.0
4	038441c925bb	038441c925bb_2018-08-18T20_30_00.jpg	4	17280.0	69840.0

Figure 6: Code snippet to split the training data into training and validation.

2.4 Model Architecture

The model we used here is Faster R-CNN, an advanced object detection model that was introduced by ShaoqingRen etc. [4]. This model is composed of three parts: convolution layers, region proposal network, and classes and bounding box prediction. The convolution layers include networks like VGG16 or ResNet, which provide feature maps that are essential for accurate object detection. Fast R-CNN follows a two-stage detection process. In the first stage, the RPN generates region proposals. In the second stage, those proposals are refined by a classifier and a bounding box regressor to produce the final object detection results. The rationale for using Fast R-CNN is that for every day, two activities, onset and wake up, can be treated as the object boundary. Since we already stored

```python
class SleepStatesDataset(torch.utils.data.Dataset):
    def __init__(self, split_df, annotations_df, images_folder, transforms):
        self.label_mapping = {'onset': 1, 'wakeup': 2}
        self.split_df = split_df
        self.annotations_df = annotations_df
        self.images_folder = images_folder
        self.transforms = transforms

    def __len__(self):
        return len(self.split_df)

    def __getitem__(self, idx):
        image_name = self.split_df['image_name'][idx]
        # print(f"image_name = {image_name}")
        image_path = os.path.join(self.images_folder, image_name)
        # print(f"image_path = {image_path}")
        image = read_image(image_path)
        window_df = self.annotations_df.loc[self.annotations_df['image_name'] == image_name]
        if len(window_df) > 0:
            boxes = torch.tensor(window_df.iloc[:, 5:].values, dtype=torch.float32)
            labels = window_df['label']
            labels = torch.tensor([self.label_mapping[l] for l in labels], dtype=torch.int64)
            area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
            iscrowd = torch.zeros(size=(len(window_df),), dtype=torch.int64)
        else:
            boxes = torch.empty(size=(0, 4), dtype=torch.float32)
            labels = torch.empty(size=(0,), dtype=torch.int64)
            area = torch.empty(size=(0,), dtype=torch.float32)
            iscrowd = torch.empty(size=(0,), dtype=torch.int64)

        image = tv_tensors.Image(image)
        # print(f"image = {image}")
        target = {}
        target['image_id'] = image_name
        target['boxes'] = tv_tensors.BoundingBoxes(boxes, format="XYXY", canvas_size=F.get_size(image))
        target['labels'] = labels
        target['area'] = area
        target['iscrowd'] = iscrowd
        # print(f"target={target}")

        if self.transforms is not None:
            image, target = self.transforms(image, target)

        return image, target
```

Figure 7: "Code snippet for obtaining the custom dataset.

whole day continues data information as the image in preprocessing, we can use this image as an target to train the model. Figure 9

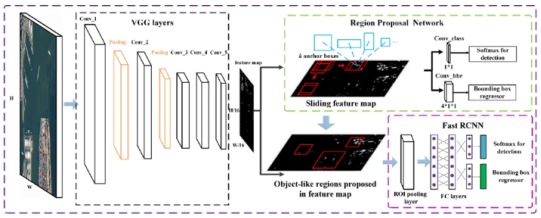

Figure 8: Faster R-CNN architecture [1]

briefly summary the model parameters we adopted.

```
In [26]: print(model)
FasterRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1440, mode='bilinear')
  )
  (backbone): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
      (bn1): FrozenBatchNorm2d(64, eps=0.0)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
```

Figure 9: The parameters used in Fast R-CNN model

2.5 Evaluation Metrics

The competition use Average Precision(AP) as the evaluation metrics, which is completely different with the classical classification problem. AP changes based on different dummy predictions in comparison to the ground-truth.

Let's take a subset of a series for this example, roughly a 48 hour period with a pair of onset and wakeup events, denoted in the chart below with the dot-slash lines. Additionally, we have the corresponding predictions for each of the 2 events in the solid lines. When the difference between the annotation and the prediction is greater than 360 steps or 30 minutes the prediction gets a score of 0 like we see below. Which means, beyond a difference of 30 minutes, predictions aren't useful in this competition (could be +30 minutes or -30 minutes).

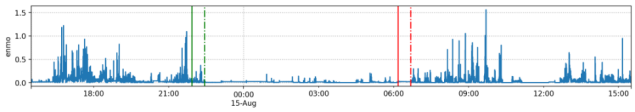

Figure 10: Difference in steps: 360 Average Precision(AP): 0.0

Next, if we reduce the error to 359 steps instead, we see a score of 0.1

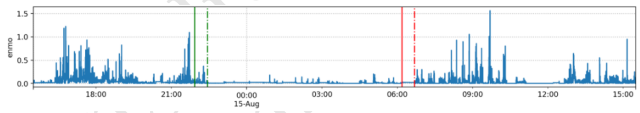

Figure 11: Difference in steps: 359 Average Precision(AP): 0.1

With a difference of 120 steps, we get a AP of 0.5

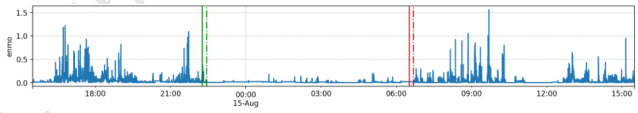

Figure 12: Difference in steps: 120 Average Precision(AP): 0.5

We reduce the error even further, taking it to just 11 steps and the AP increases to 1.0

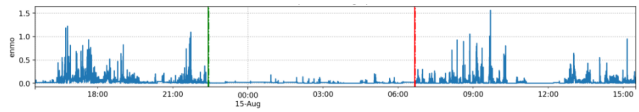

Figure 13: Difference in steps: 11 Average Precision(AP): 0.9

3 RESULT

In the evaluation phase, we employed the CocoEvaluator class to systematically assess the model's performance across various Intersection over Union (IoU) types. Key to this process is the versatility of the CocoEvaluator, which adeptly handles different IoU types — 'bbox' for bounding box detection, 'segm' for segmentation, and 'keypoints' for keypoint detection. Each type necessitates a unique approach to preparing and processing the data, and the code addresses these requirements with precision. The evaluation results are shown in Table 3.

In addition to assessing the performance of the model using average recall, we also inspect real images to observe the final results. We present two typical results from our evaluation process.

Table 3: Performance of Faster R-CNN in child sleep state detection

Epoch	Average Recall(Training)	Average Recall(Validation)
1	0.667	0.6257
2	0.764	0.6861
3	0.770	0.7026
4	0.778	0.7106
5	0.781	0.7113
6	0.782	0.7146
7	0.780	0.7122
8	0.781	0.7135
9	0.781	0.7125
10	0.780	0.713

- Examining a selected images that don't have any annotations

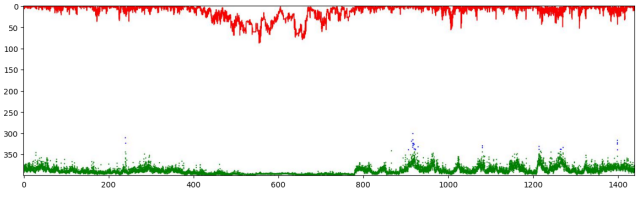

Figure 14: A selected image that doesn't have any annotations

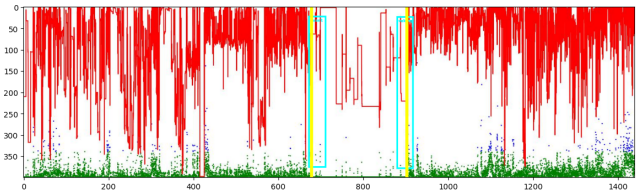

Figure 15: A predict image for the selected image

- Examining a selected images that have exactly 2 annotations

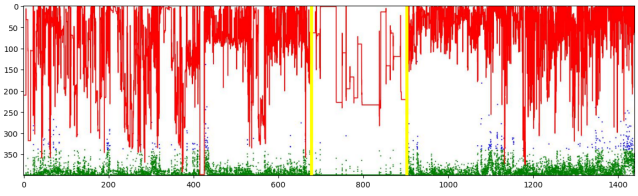

Figure 16: A selected image that has two annotations

In these figures, the green line indicates the actual onset and wake-up activity based on the annotation information, while the yellow box indicates the possible onset and wake-up activity. From those

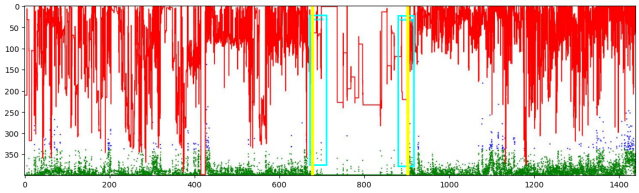

Figure 17: A predict image for the selected image

images, We can see clearly that if the selected images don't have any annotations, after training, for a new image that may have the same attributes, the model will predict that there are no onset and wake-up activities. However, when the selected images have one or two annotations, our model could detect the potential activity onset and wake-up.

4 CONCLUSION AND FUTURE WORK

In this project, our team implemented a Faster R-CNN to detect child sleep states using images cut off from the wrist-worn accelerometer data. The average recall is 0.698 in the training set, while achieving 0.6361 in the validation set, which is better when compared to the published performance of other models. Since the organization didn't provide much data for this competition before the deadline, we couldn't obtain more data to train the model. However, based on the performance on this limited data, we believe Faster R-CNN is a good option for detecting sleep states in children, considering its generalization potential and widespread use in the future.

In the future, if the organization releases more data for this dataset, we will try to optimize our model and gain a deeper understanding of object detection models.

ACKNOWLEDGMENTS

We are indebted to Professor Yan Yan, who taught CS577, a course on Deep Learning at the Department of Computer Science at the Illinois Institute of Technology. Under his guidance, we gained a comprehensive understanding of deep learning algorithms and the implementation of their underlying principles. Professor Yan's expertise and instruction were invaluable in deepening our knowledge and skills in this field.

REFERENCES

[1] Zhipeng Deng, Hao Sun, Shilin Zhou, Juanping Zhao, Lin Lei, and Huanxin Zou. 2018. Multi-scale object detection in remote sensing imagery with convolutional neural networks. *ISPRS journal of photogrammetry and remote sensing* 145 (2018), 3–22.

[2] Alexandre Domingues, Teresa Paiva, and J Miguel Sanches. 2013. Sleep and wakefulness state detection in nocturnal actigraphy based on movement information. *IEEE Transactions on Biomedical Engineering* 61, 2 (2013), 426–434.

[3] Lisa J Meltzer, Hawley E Montgomery-Downs, Salvatore P Insana, and Colleen M Walsh. 2012. Use of actigraphy for assessment in pediatric sleep research. *Sleep medicine reviews* 16, 5 (2012), 463–475.

[4] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28 (2015).

[5] Guto Leoni Santos, Patricia Takako Endo, Kayo Henrique de Carvalho Monteiro, Elisson da Silva Rocha, Ivanovitch Silva, and Theo Lynn. 2019. Accelerometer-based human fall detection using convolutional neural networks. *Sensors* 19, 7 (2019), 1644.

[6] Edward Sazonov, Nadezhda Sazonova, Stephanie Schuckers, Michael Neuman, CHIME Study Group, et al. 2004. Activity-based sleep–wake identification in

infants. *Physiological measurement* 25, 5 (2004), 1291.
[7] Huan Song, Deepta Rajan, Jayaraman Thiagarajan, and Andreas Spanias. 2018. Attend and diagnose: Clinical time series analysis using attention models. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
[8] Kalaivani Sundararajan, Sonja Georgievska, Bart HW Te Lindert, Philip R Gehrman, Jennifer Ramautar, Diego R Mazzotti, Séverine Sabia, Michael N Weedon, Eus JW van Someren, Lars Ridder, et al. 2021. Sleep classification from wrist-worn accelerometer data using random forests. *Scientific reports* 11, 1 (2021), 24.
[9] Olivia Walch, Yitong Huang, Daniel Forger, and Cathy Goldstein. 2019. Sleep stage prediction with raw acceleration and photoplethysmography heart rate data derived from a consumer wearable device. *Sleep* 42, 12 (2019), zsz180.
[10] Zhiguang Wang, Weizhong Yan, and Tim Oates. 2017. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*. IEEE, 1578–1585.
[11] Kun Xia, Jianguang Huang, and Hanyu Wang. 2020. LSTM-CNN architecture for human activity recognition. *IEEE Access* 8 (2020), 56855–56866.

A WEBSITE LINKS

- https://www.kaggle.com/competitions/child-mind-institute-detect-sleep-states
- https://github.com/BOBWang1117/Child-Sleep-State-Detection-in-Faster-R-CNN-Model
- https://www.kaggle.com/competitions/child-mind-institute-detect-sleep-states/data

B README FILE

Name: CS577_Deep-Learning_Project

Team Member:

- Xue Zhang
- Hongbo Wang

Environment:

- Anaconda 23.7.4
- Python 3.11.5

Details:

- Type: team project
- Professor: Yan Yan
- Developing Language: Python
- Project Name: Child Sleep State Detection from Wrist-Worn Accelerometer Data Using A Faster R-CNN Model
- Time: Dec/1/2023

Dependencies:

- Kaggle data set: data set

Install package:

- `troch`: 2.1.0
- `pandas`: 2.0.3
- `numpy`: 1.24.3
- `tdmq`: 4.65.0
- `faster_rcnn`: 4.65.0

Run Program:

- Download Anaconda and necessary environment
- Download the data set
- Run faster-rcnn-final.ipynb file

C SOURCE CODE

CS577_Deep-Learning_Project
https://github.com/BOBWang1117/Child-Sleep-State-Detection-in-Faster-R-CNN-Model