Paper reproduction: Bringing At-home Pediatric Sleep Apnea Tesing Closer to Reality: A Multi-Modal Transformer Approach

Aaron Schlesinger, Dave Pankros

May 02, 2023

1 Paper reproduction: Bringing At-home Pediatric Sleep Apnea Tesing Closer to Reality: A Multi-Modal Transformer Approach

Aaron Schlesinger and Dave Pankros
{aschle2, pankros2}@illinois.edu
Team ID: 3

This notebook is the final submission of our paper reproduction for CS 598 - Deep Learning for healthcare. In it, we aim to reproduce the findings in the paper *Bringing At-home Pediatric Sleep Apnea Testing Closer to Reality: A Multi-Modal Transformer Approach* [1].

The code herein is functional, with some limitations around data, discussed below. Our public GitHub repository in which we did our work and from which we derived this notebook is located at github.com/arschles/UIUC-CS598-DLH-Project. The code therein is organized as a standard Python project, and the repository contains comprehensive instructions on running it.

Our accompanying introductory video for this project is located at drive.google.com/file/d/1-26rkxSZ0gM0GsEHTd0FOJs5jWnF9Pt0/view?usp=sharing.

2 License

The data used by this dataset is subject to a license. While we are not sharing the original data in its raw form, we are sharing derivations of that data like pre-trained model checkpoints. To facilitate the goals of this assignment and class and to honor the spirit of the license, we require that by opening and running this notebook you agree that you:

1. Will not attempt to reverse engineer the data into a form other than provided,
2. Will not attempt to identify or de-anonymize any individual or institution in the dataset,
3. Will not share or re-use the data in any form,
4. Will not use the data for any purpose other than this assignment, and
5. Maintain a up-to-date cerification in human research subject protection and HIPAA regulations.

The data license is discussed in more detail below.

3 Preface

Much effort went into making the original paper repository runnable. The repository was lacking any documentation including even python version and package versions. It was built exclusively to be run on Windows and lacked any attempt at cross-platform support. We ran into several cases where the code, as supplied, could never have been run in the provided state. In one instance, for example, the arguments to a function were illegal and, after reaching out to the author and receiving no reply, we used our best judgment at a solution.

With that in mind, we are attempting to reproduce results consistent with the original paper, but there may be differences due to these changes or other instances of errors or omissions that did not cause a code failure and that may have been too subtle to be caught in our initial passes over the code. These differences will, inevitably, cause deviations between our results and the results presented in the paper.

We have, however, undertaken in our fork of the original code to provide more information to aid reproducibility (including a `requirements.txt` file and other detailed dependency information), made the code cross-platform where it was not, and provide for information about how to preprocess and run the code using standard tools like bash and make.

4 Introduction

Sleep apnea in children is a major health problem that affects between one to five percent of US children, but differs from sleep apnea in adults in its clinical causes and characteristics. Thus, previously-created methods for detecting adult sleep apnea may be ineffective at detecting pediatric sleep apnea.

4.1 Background

While there are numerous testing tools and algorithmic methods for detecting adult sleep apnea, the same tools and methods are unavailable for pediatric sleep apnea (PSA) due to these differences. Detecting pediatric sleep apnea more quickly and easily can lead to earlier clinical intervention in the child's care and ultimately prevent the wide variety of health issues commonly caused by Obstructive Sleep Apnea (OSA). These effects depend on the underlying cause of the OSA, but can include poor attention span, hyperactivity, poor school performance, craniofacial development abnormalities, narrow palate, dentition issues, increased risk of cavities, cognitive deficits, poor quality of life, elevated blood pressure, elevated inflammatory markers, to name a few. [2,3] This list does not include the secondary effects of misdiagnosis of PSA and the medication-induced side effects caused by its misdiagnosis, which is unfortunately common. It is, therefore, extremely important to identify and provide early intervention to treat Pediatric OSA before the worst of these symptoms occur.

Polysomnography (PSG) is the standard method for formal sleep apnea diagnosis, but is generally performed in a dedicated facility where a patient can be monitored overnight. Polysomnography involves collecting various continuous-time signals, including electroencephalogram (EEG), electrooculogram (EOG), electrocardiogram (ECG), pulse oximetry (SpO2), end-tidal carbon dioxide (ETCO2), respiratory inductance plethysmography (RIP), nasal airflow, and oral airflow. While effective, PSG is, however, complex, costly and requires a dedicated sleep lab. That a sleep lab is required makes OSA in children inaccessible to under-served areas and economically disadvantaged

populations. If OSA can be detected earlier and with inexpensive and commonly-available equipment the most-severe impacts can be lessened leading to better outcomes and a lower cost to the healthcare system generally.

4.1.1 State of the Art

Current methods target adults and, for reasons stated earlier, are ineffective at diagnosing PSA in children. Very little work has been done in the scope of pediatric sleep apnea (PSA). In general, full Polysomnography data is hard to find and thus, much research has focused on determining the Apnea-Hypopnea Index (AHI) from ECG and SpO2 signals.

While transformers are used commonly in general deep learning models, they are much less prevalent in the detection of sleep apnea. Two studies described in this paper used transformers to determine sleep stages (one in adults, one in children), while another used a hybrid CNN/transformer model of obstructive sleep apnea (OSA) detection.

4.2 Paper

The paper proposes to study the gaps in Obstructive Sleep Apnea Hypopnea Syndrome (OSAHS) in children vis-a-vis adults. The paper then suggests a custom transformer-based method and data representation for PSA detection, and identifies the polysomography modalities that most closely correlate to OSAHS in children.

The results presented in the paper portray state-of-the-art results.

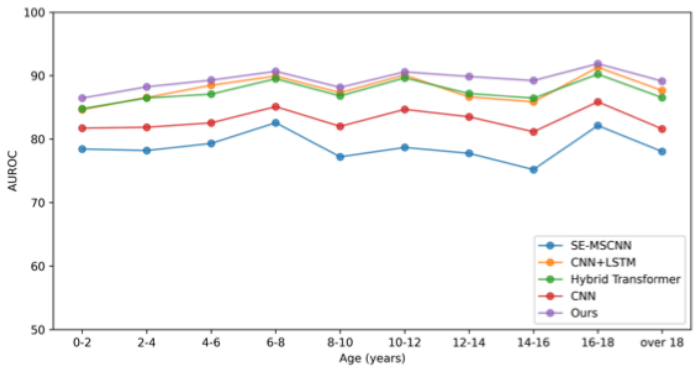


Figure 2: Performance comparison of the proposed and state-of-the-art models on different age groups measured by AUROC on NCH dataset.

5 Scope of Reproducibility

We will generally investigate whether we can beat state-of-the-art results from Polysomnography (PSG) studies in pediatric sleep apnea detection using the transformer-based model proposed in the paper and discussed above. More specifically, we will focus on testing the following hypotheses:

1. Whether the proposed model can achieve results from signals more easily collected than PSG. As in the paper, we will focus on ECG and SpO_2 signals, and
2. Whether the results support this method being effective as a dedicated method for in-lab sleep studies.

6 Methodology

In this section, we detail how we acquire, process and load the relevant data, and how we train and evaluate the model given those data. We will exlain the procedure and code in this notebook with the understanding that we did not process the data, train the model, or evaluate the model using this notebook. True reproduction of our environment is explained in the Repository Setup and Evaluation section, immediately following.

6.1 Environment

The original project lacked any definition of python and dependency version requirements. We documented the complete list of python packages and versions necessary to run our repository in our environment.yaml file. For convenience, the primary requirements we chose were: - python 3.11 - tensorflow 2.15 - keras 2.16 - numpy 1.26 - pandas 2.2 - scipy 1.12

7 Repository Setup

We originally started with the original research code, copied it into this project's repository, and began making changes. As detailed elsewhere, much work was done to get the code to a runnable state, then more was done to make it performant, able to load models from checkpoints, and so on.

When we got to a good state in the repository, we moved code into this notebook and modified it as necessary to work in this environment. Some of those modifications include the following:

- Intra-project imports don't work so we had to remove them and linearize the intra-project dependency graph herein
- Everything is in a global namespace, so we had to fix the naming conflicts that arose herein
- Testing and training code is combined in the repository, but it made more sense to split the two pieces herein, so we had to extract large segments of code common to both operations

Despite the differences between code in this notebook and code in the repository, we achieve the same result in both.

If you'd like to run the code in the repository, please see the `original/HOW_TO_RUN.md` file

7.1 Setup and configuration

First, we provide some package installation, setup and configuration code. These configuration settings will be valid for this section as well as the remainder of the notebook.

The values in this block should be reviewed and updated to ensure they will work in the environment in which you will be executing them.

[4]:
```
!pip install 'numpy~=1.26.4' 'pandas~=2.2.1' 'scipy~=1.12.0' 'mne~=1.6.1'␣
↪'keras<2.16' 'tensorflow~=2.15.1' 'scikit-learn~=1.4.1.post1'␣
↪'python-dateutil~=2.8.2' 'pyright' 'pylint' 'matplotlib' 'tensorflow-addons'␣
↪'biosppy~=2.1.2'
```

```
Requirement already satisfied: numpy~=1.26.4 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (1.26.4)
Requirement already satisfied: pandas~=2.2.1 in
```

```
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (2.2.1)
Requirement already satisfied: scipy~=1.12.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (1.12.0)
Requirement already satisfied: mne~=1.6.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (1.6.1)
Requirement already satisfied: keras<2.16 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (2.15.0)
Requirement already satisfied: tensorflow~=2.15.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (2.15.1)
Requirement already satisfied: scikit-learn~=1.4.1.post1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (1.4.1.post1)
Requirement already satisfied: python-dateutil~=2.8.2 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (2.8.2)
Requirement already satisfied: pyright in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (1.1.361)
Requirement already satisfied: pylint in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (3.1.0)
Requirement already satisfied: matplotlib in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (3.8.3)
Requirement already satisfied: tensorflow-addons in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (0.23.0)
Requirement already satisfied: biosppy~=2.1.2 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (2.1.2)
Requirement already satisfied: pytz>=2020.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pandas~=2.2.1)
(2024.1)
Requirement already satisfied: tzdata>=2022.7 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pandas~=2.2.1)
(2024.1)
Requirement already satisfied: tqdm in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from mne~=1.6.1)
(4.66.2)
Requirement already satisfied: pooch>=1.5 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from mne~=1.6.1)
(1.8.1)
Requirement already satisfied: decorator in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from mne~=1.6.1)
(5.1.1)
Requirement already satisfied: packaging in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from mne~=1.6.1)
(24.0)
Requirement already satisfied: jinja2 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from mne~=1.6.1)
(3.1.3)
Requirement already satisfied: lazy-loader>=0.3 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from mne~=1.6.1)
(0.3)
Requirement already satisfied: absl-py>=1.0.0 in
```

```
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (3.10.0)
Requirement already satisfied: libclang>=13.0.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.3.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (0.3.2)
Requirement already satisfied: opt-einsum>=2.3.2 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (3.3.0)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in /root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (4.24.4)
Requirement already satisfied: setuptools in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (69.2.0)
Requirement already satisfied: six>=1.12.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (4.10.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (0.36.0)
```

```
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (1.59.3)
Requirement already satisfied: tensorboard<2.16,>=2.15 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorflow~=2.15.1) (2.15.0)
Requirement already satisfied: joblib>=1.2.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from scikit-
learn~=1.4.1.post1) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from scikit-
learn~=1.4.1.post1) (3.4.0)
Requirement already satisfied: nodeenv>=1.6.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pyright)
(1.8.0)
Requirement already satisfied: platformdirs>=2.2.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pylint) (4.2.0)
Requirement already satisfied: astroid<=3.2.0-dev0,>=3.1.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pylint) (3.1.0)
Requirement already satisfied: isort!=5.13.0,<6,>=4.2.5 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pylint)
(5.13.2)
Requirement already satisfied: mccabe<0.8,>=0.6 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pylint) (0.7.0)
Requirement already satisfied: tomlkit>=0.10.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pylint)
(0.12.4)
Requirement already satisfied: dill>=0.3.6 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from pylint) (0.3.8)
Requirement already satisfied: contourpy>=1.0.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from matplotlib)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from matplotlib)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from matplotlib)
(4.50.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from matplotlib)
(1.4.5)
Requirement already satisfied: pillow>=8 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from matplotlib)
(10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
```

```
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from matplotlib)
(3.1.2)
Requirement already satisfied: typeguard<3.0.0,>=2.7 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from tensorflow-
addons) (2.13.3)
Requirement already satisfied: bidict in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from biosppy~=2.1.2)
(0.23.1)
Requirement already satisfied: shortuuid in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from biosppy~=2.1.2)
(1.0.13)
Requirement already satisfied: opencv-python in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from biosppy~=2.1.2)
(4.9.0.80)
Requirement already satisfied: pywavelets in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from biosppy~=2.1.2)
(1.4.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
astunparse>=1.6.0->tensorflow~=2.15.1) (0.43.0)
Requirement already satisfied: requests>=2.19.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
pooch>=1.5->mne~=1.6.1) (2.31.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (2.29.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (3.6)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (0.7.0)
Requirement already satisfied: werkzeug>=1.0.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (3.0.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
jinja2->mne~=1.6.1) (2.1.5)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from google-
auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from google-
auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in
```

```
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from google-
auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from google-auth-
oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (2.0.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
requests>=2.19.0->pooch>=1.5->mne~=1.6.1) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
requests>=2.19.0->pooch>=1.5->mne~=1.6.1) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
requests>=2.19.0->pooch>=1.5->mne~=1.6.1) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
requests>=2.19.0->pooch>=1.5->mne~=1.6.1) (2024.2.2)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from
pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (0.5.1)
Requirement already satisfied: oauthlib>=3.0.0 in
/root/miniconda3/envs/dlhproj/lib/python3.11/site-packages (from requests-
oauthlib>=0.7.0->google-auth-
oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow~=2.15.1) (3.2.2)
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

[5]:
```python
import os

#####
# Preprocessing
#####

# raw data is required for both the preprocessing and data loading step.
# since raw data is licensed, we do not, in this notebook, do these steps
# by default.
#
# if you have access to raw data, download it to your machine, set this
# variable to True and set PHYSIONET_ROOT to the location on disk of your data's
# physionet.org directory
SHOULD_PREPROCESS = False
PHYSIONET_ROOT = "/tmp/data/physionet.org"
```

```
# We have preprocessed the data and hosted it at the below URL. Do not change␣
 ↪this value.
#
# Data will remain present at this URL through May 31, 2024
PREPROCESS_URL = "https://dlhproject.sfo3.cdn.digitaloceanspaces.com/nch_30x64.
 ↪npz"

#####
# Training
#####

# this is handy to turn off the training process. it should only be
# set to true if TEST_FROM_CHECKPOINT is also set to True
SKIP_TRAINING = False

# when training, how many epochs should be run?  This can be as high as 200,␣
 ↪but you must have some sort of tensor processor for that to be feasible
NUM_EPOCHS = 2  # 2 is good for a notebook demonstration (training is around 3m␣
 ↪on an M1 Mac), but provides mediocre results.  25 is much better.  100 and␣
 ↪higher will be excellent, if you have the hardware and time for it

# The model to train.  "Transformer" "is the model from this paper.  This is␣
 ↪probably the one you want.
MODEL = "Transformer"

#####
# Testing
#####

# Flag to determine whether to test from a pretrained model checkpoint.
# set to False if you want to test from the model that is trained in this␣
 ↪notebook,
# or True if you want to download the checkpoint and test with that.
#
# NOTE: the model will train regardless of the value to which you set this
# constant. it only affects how testing is done
#
# If you want to skip training, set the SKIP_TRAINING flag below to True
TEST_FROM_CHECKPOINT = True
CHECKPOINT_URL = "https://dlhproject.sfo3.cdn.digitaloceanspaces.com/
 ↪Transformer_SPO2ECG_200.zip"

#####
# do not change anything under this line
#####
```

```
if SKIP_TRAINING and not TEST_FROM_CHECKPOINT:
    print(
        "WARNING: SKIP_TRAINING is on and TEST_FROM_CHECKPOINT is off. "
        "This configuration will result a completely untrained model, and your "
        "evaluation metrics will probably be horrible."
    )

AHI_OUT_PATH = os.path.join(PHYSIONET_ROOT, 'AHI.csv')
PREPROCESS_OUT_PATH = os.path.join(PHYSIONET_ROOT, 'nch_30x64.npz')
NCH_DATA_ROOT = os.path.join(PHYSIONET_ROOT, 'files', 'nch-sleep', '3.1.0')
HEALTH_DATA_ROOT = os.path.join(NCH_DATA_ROOT, 'Health_Data')
SLEEP_DATA_ROOT = os.path.join(NCH_DATA_ROOT, 'Sleep_Data')
MODEL_DIR = os.path.join('original', 'weights')
MODEL_OUT_PATH = os.path.join(MODEL_DIR, f"{MODEL}_NOTEBOOK_{NUM_EPOCHS}")
```

7.2 Data

7.2.1 Data Licensing Note

Due to the license required to access the data, we cannot provide the raw dataset with this notebook because doing so would be a violation of terms 3 and 4 of the PhysioNetCredential Health Data License 1.5.0.

Because we cannot supply original raw data, we will discuss the processing of the data and the relevant processing code with example output, but the provided code **will not actually process data** within this notebook. Since it is not subject to the aforementioned license, we have stored preprocessed data elsewhere and will load it in relevant sections of this notebook. From that point, further processing will be performed in this notebook.

> Before we proceed, we want to stress that this preprocessed data – and the model checkpoint data we'll discuss later – is derived from licensed data for which one must pass a training course to access. This notebook includes instructions and code to access these resources. **By proceeding past this section, you must agree to adhere to the requirements outlined in the license section above**.

7.2.2 Source(s)

The original paper used Nationwide Children's Hospital (NCH) Sleep Data Bank (Lee et al., 2022), and Childhood Adenotonsillectomy Trial (CHAT) dataset (Marcus et al., 2013; Redline et al., 2011). Each of these datasets are collected from actual sleep studies, anonymized and made available for research. We were granted access to the CHAT dataset very late in timeline of this project. We considered incorporating it into our code, but the lateness coupled with the state of the code in other sections implied a high of risk that we were not able to align with the project timeline. Therefore, we decided to forego the CHAT dataset in favor of using the NCH dataset only.

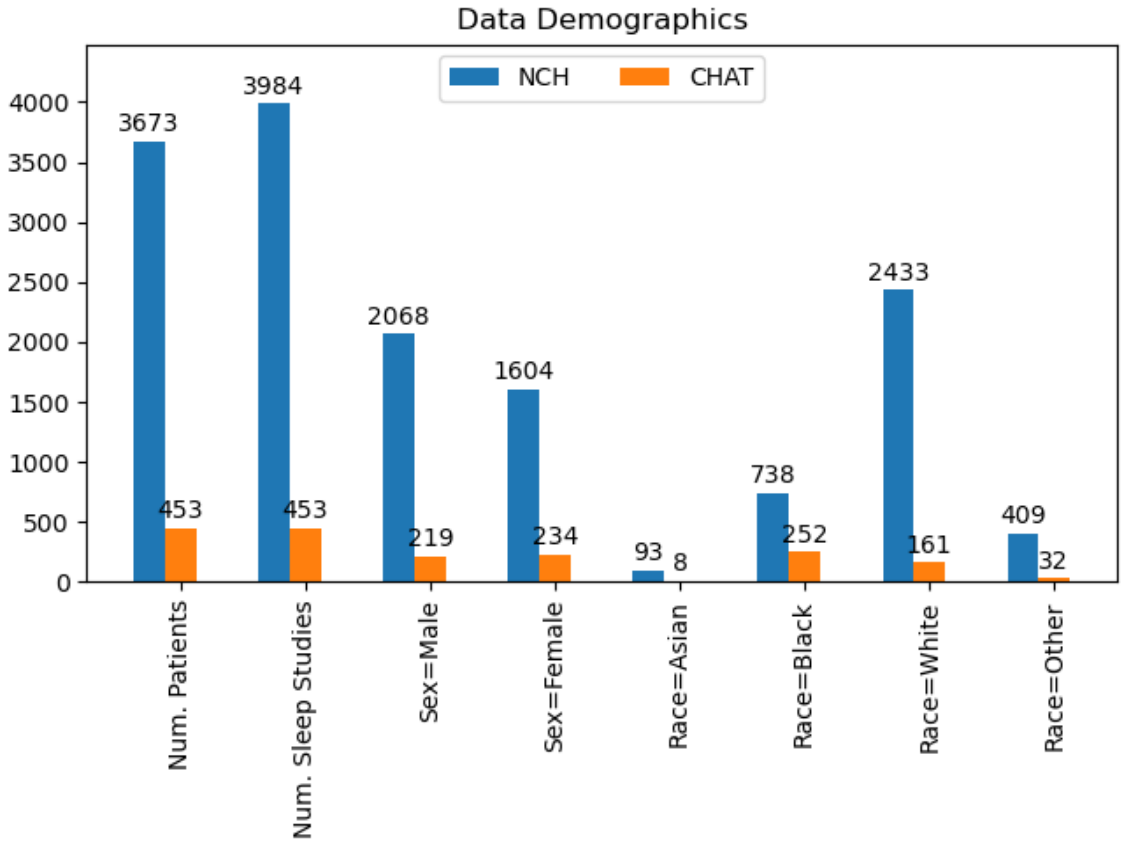
We currently have access to the NCH dataset through physionet.org. We downloaded the dataset for over a month and were able to obtain about half of the overall content because all download speeds were capped at around 600KB/s. As described later, we only used about half of the downloaded dataset (about 25% of the overall dataset) due to resource constraints in processing the data.

7.3 Dataset summary

Since the paper provides extensive statistics on the data used in this study, we have not undertaken to perform our own calculations. The paper-provided measures are illustrated below.

7.3.1 Demographics of the Datasets

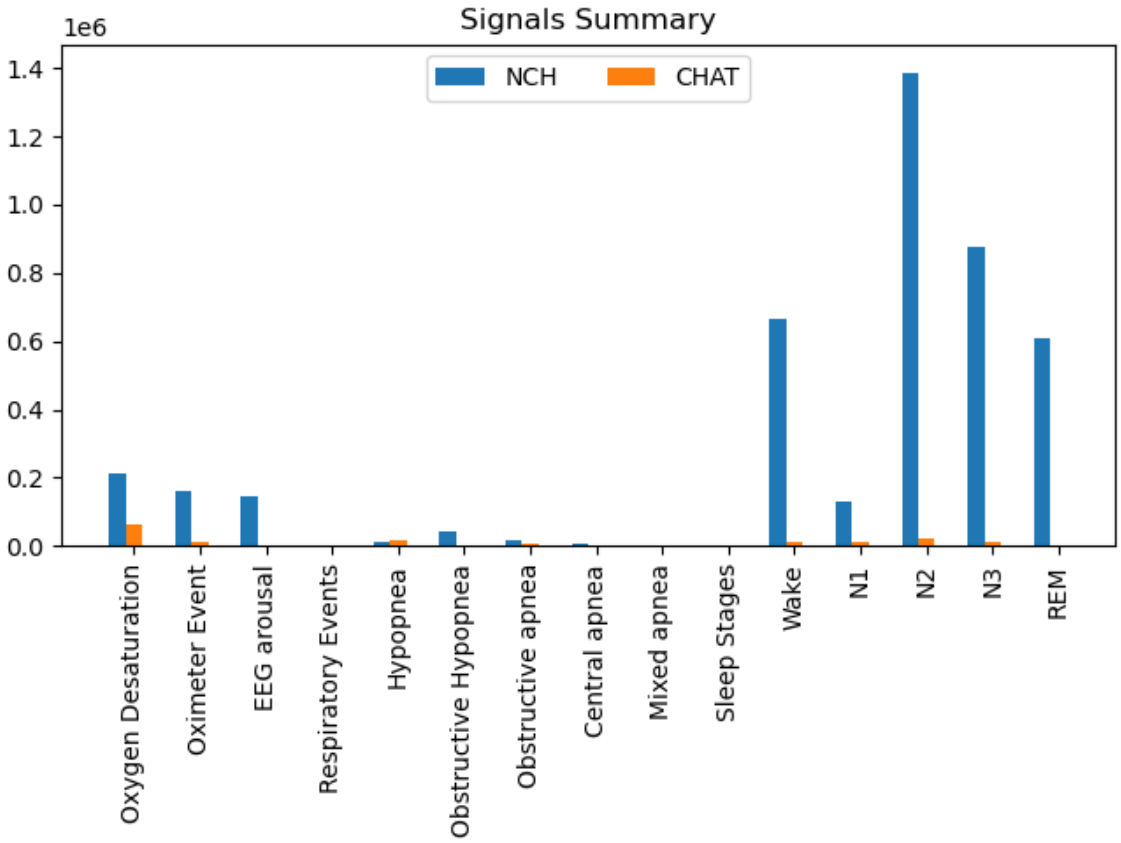
The dataset are summarized in this section by patient demographics, in charted and tabular format.



		NCH	CHAT
	Number of Patients	3673	453
	Number of Sleep Studies	3984	453
Sex			
	Male	2068	219
	Female	1604	234
Race			
	Asian	93	8
	Black	738	252
	White	2433	161
	Other	409	32
	Age (years/mean)	[0-30]/8.8	[5-9]/6.5

7.3.2 Data Statistics

The data are summarized in this section by the various collected signals, in both charted and tabular format.



Event	NCH	CHAT
Oxygen Desaturation	215280	65006
Oximeter Event	161641	9,864
EEG arousal	146052	–
Respiratory Events		
Hypopnea	14522	15871
Obstructive Hypopnea	42179	–
Obstructive apnea	15782	7075
Central apnea	6938	3656
Mixed apnea	2650	–
Sleep Stages		
Wake	665676	10282
N1	128410	13578
N2	1383765	19985
N3	875486	9981
REM	611320	3283

7.4 Data preprocessing

7.4.1 Data Normalization: Interpolation, Resampling and Tokenization

The raw data are categories roughly as follows:

1. Regular time-series.
2. irregular time-teries, and
3. tabular data.

Additionally, the time-series data may be provided in various frequencies. To merge all the different data types into a coherent dataset suitable for training and testing, the following steps must be performed:

1. Each irregular time series must be interpolated to convert it into a regular time series.
2. All the time series data must be resampled into a uniform frequency, $f_{sampling}$, for all sleep studies and modalities.
3. The tabular data is added to the time series as a constant signal (i.e. repeated tokens)
4. The combined data is split into i equal-length tokens of time S, where each modality consists of $S * f_{sampling}$ data points

These data can then be split and passed to the model for training and testing.

7.4.2 Splitting

This paper utilizes a custom stratified k-fold cross validation to achieve the following:

1. Ensure that an equal number of patients are assigned to each fold, and
2. normalize the number of positive samples in each fold.

Pseudocode for this method is as follows:

Input: $\{P_n\}_{n=1}^{N}$
Output: $\{fold_f\}_{n=1}^{K}$
for $n \leftarrow 1$ ***to*** N **do**
    $P_i.score = 0$
    for $m \leftarrow 1$ ***to*** M_n **do**
        for $l \leftarrow 1$ ***to*** $L_{n,m}$ **do**
            $P_i.score \mathrel{+}= length(E_{i,j}^{k})$
        end
    end
end
score_sorted_list $\leftarrow$ sort patients by score
for $i \leftarrow 1$ ***to*** N **do**
    $f = i \mod K$
    $fold_f \leftarrow$ score_sorted_list[i]
end

7.4.3 The Apnea-Hypopnea index

The Apnea-Hypopnea index (AHI) is an important quantification of the severity of sleep apnea. Its derivation for a single night of sleep is as follows:

$$\frac{N_{apneic} + N_{hypopneic}}{H}$$

Where the following are defined:

- N_{apneic} is the number of apneic events (instances where the patient stops breathing),
- $N_{hypopneic}$ is the number of hypopneic events (instances where the airflow is blocked and the patient's breathing becomes more shallow), and
- H is the total number of hours the patient slept that night

Since it can be derived from an entire night of sleep, the AHI is a very useful measure to summarize, with relatively modest loss of information, a person's apnea/hypopnea activity in a given night of

sleep.

Below we show some setup code followed by the code to create a TSV (tab-separated file) file containing AHI measures for a given sleep study.

AHI Calculation Next, after setup code is complete, we compute AHI values.

[6]:
```
import os
import os.path
import pandas as pd
from datetime import datetime
import csv

APNEA_EVENT_DICT = {
    "Obstructive Apnea": 2,
    "Central Apnea": 2,
    "Mixed Apnea": 2,
    "apnea": 2,
    "obstructive apnea": 2,
    "central apnea": 2,
    "apnea": 2,
    "Apnea": 2,
}

HYPOPNEA_EVENT_DICT = {
    "Obstructive Hypopnea": 1,
    "Hypopnea": 1,
    "hypopnea": 1,
    "Mixed Hypopnea": 1,
    "Central Hypopnea": 1,
}


def _num_sleep_hours(
        sleep_study_metadata: pd.DataFrame,
        pat_id: int,
        study_id: int,
) -> int:
    ssm = sleep_study_metadata
    sleep_duration_df = ssm.loc[
        (ssm["STUDY_PAT_ID"] == pat_id) &
        (ssm["SLEEP_STUDY_ID"] == study_id)
        ]
    assert len(sleep_duration_df) == 1, (
        f'expected just 1 study with patient {pat_id} and study '
        f'{study_id}, but got {len(sleep_duration_df)} instead'
    )
    sleep_duration_datetime = datetime.strptime(
```

```
        str(
            sleep_duration_df[
                "SLEEP_STUDY_DURATION_DATETIME"
            ].iloc[0]
        ).strip(),
        "%H:%M:%S"
    )
    return sleep_duration_datetime.hour


def _ahi_for_study(
        sleep_study_metadata: pd.DataFrame,
        sleep_study: pd.DataFrame,
        pat_id: int,
        study_id: int,
) -> float:
    """
    Calculate the apnea-hypopnea index (AHI) for a given sleep study.
    All apnea and hypopnea events will be counted from the sleep_study
    DataFrame, and then divided by the total sleep duration, which
    will be fetched from the sleep_study_metadata DataFrame. The result will
    be returned as a float.

    For more on AHI, see the following link:

    https://www.sleepfoundation.org/sleep-apnea/ahi

    :param sleep_study_metadata
        the DataFrame that has at least the following columns in order
        from left to right:
        STUDY_PAT_ID,
        SLEEP_STUDY_ID,
        SLEEP_STUDY_START_DATETIME,
        SLEEP_STUDY_DURATION_DATETIME
    :param sleep_study
        the data from the sleep study in which we're interested
    :param pat_id
        the ID of the patient on whom the given study was done
    :param study_id
        the ID of the study

    :return
        the AHI for the given study, as a float value
    """

    # example tab-separated (TSV) file:
    #
```

```
    # onset duration description
    # 29766.7421875        11.0546875        Obstructive Hypopnea

    df = sleep_study
    hypopnea_keys = set(HYPOPNEA_EVENT_DICT.keys())
    apnea_keys = set(APNEA_EVENT_DICT.keys())

    hypopnea_events = df.loc[df["description"].isin(hypopnea_keys)]
    apnea_events = df.loc[df["description"].isin(apnea_keys)]
    total_num_events = len(hypopnea_events) + len(apnea_events)
    sleep_hours = float(_num_sleep_hours(
        sleep_study_metadata,
        pat_id,
        study_id,
    ))
    return float(total_num_events) / sleep_hours


def _parse_ss_tsv_filename(filename: str) -> tuple[int, int]:
    '''
    given a sleep study filename like `10048_24622.tsv`, that represents
    <patient_id>_<sleep_study_id>.tsv, return a 2-tuple containing
    the patient ID in element 1 and sleep study ID in element 2
    '''
    if not filename.endswith(".tsv"):
        raise FileNotFoundError(
            f"expected {filename} to end with .tsv but it didn't"
        )

    underscore_spl = filename.split("/")[-1][:-4].split("_")
    if len(underscore_spl) != 2:
        raise FileNotFoundError(f'malformed filename {filename}')
    [pat_id, study_id] = underscore_spl
    return (int(pat_id), int(study_id))


def _write_tsv(out_filename: str, data: list[tuple[str, str, float]]):
    with open(out_filename, 'w', newline='') as tsvfile:
        writer = csv.writer(tsvfile, delimiter=',')
        # in ./preprocessing.py, we need to have at least 'Study'
        # and 'AHI'. Since they chose PascalCase, I extended that usage
        # to patient ID.
        writer.writerow(("PatID", "Study", "AHI"))
        for row in data:
            writer.writerow(row)
```

```
def calculate_ahi(
        sleep_study_metadata_file: str,
        sleep_study_root: str,
        out_file: str,
) -> None:
    '''
    Calculate the AHI values from a sleep study's metadata file and all the
    sleep measurements in a given directory. See the _ahi_for_study function
    for an overview of how the metadata file and sleep measurement files should
    be structured.

    :param sleep_study_metadata_file - the metadata file summarizing the sleep
        study
    :param sleep_study_root - the root directory containing all the individual
        sleep study measures

    :return out_file - the name of the file to which to write the AHI values
        in TSV format

    '''
    metadata_df = pd.read_csv(
        sleep_study_metadata_file,
        sep=","
    )

    tsv_files = [
        f for f in os.listdir(sleep_study_root)
        if f.endswith(".tsv")
    ]

    print(
        f"creating AHI from {len(tsv_files)} sleep studies in "
        f"{sleep_study_root} and outputting the AHI results to {out_file}"
    )

    # each tuple is (patient_id, study_id, AHI)
    results: list[tuple[str, str, float]] = []
    for tsv_file in tsv_files:
        filename = os.path.join(sleep_study_root, tsv_file)
        pat_id, study_id = _parse_ss_tsv_filename(filename)
        sleep_study_df = pd.read_csv(
            filename,
            sep="\t",
        )
        ahi = _ahi_for_study(
            metadata_df,
            sleep_study_df,
```

```
            pat_id,
            study_id,
        )
        results.append((pat_id, study_id, ahi))
    _write_tsv(out_file, results)


def generate_ahi_file(data_root: str, out_file: str) -> None:
    sleep_study_metadata_file = os.path.join(
        data_root,
        "files",
        "nch-sleep",
        "3.1.0",
        "Health_Data",
        "SLEEP_STUDY.csv"
    )
    sleep_study_root = os.path.join(
        data_root,
        "files",
        "nch-sleep",
        "3.1.0",
        "Sleep_Data"
    )

    calculate_ahi(
        sleep_study_metadata_file,
        sleep_study_root,
        out_file=out_file,
    )


if SHOULD_PREPROCESS:
    print(
        f"SHOULD_PREPROCESS is true, so generating AHI file from data at "
        f"{PHYSIONET_ROOT} and outputting to {AHI_OUT_PATH}"
    )
    generate_ahi_file(PHYSIONET_ROOT, AHI_OUT_PATH)
```

7.4.4 Preprocessing code

With the AHI values calculated and saved to a TSV file, we can move onto data preprocessing. Roughly speaking, the goals of preprocessing are as follows for each sleep study:

- Filter out all studies for patients whose AHI is lower than a threshold
- Collate and pad (as necessary) the relevant sleep events in the study
- Resample samples from raw data as necessary
  - The motivation behind, and method for Resampling was discussed above
- Write results to a compressed numpy representation

Sleep study loading and manipulation logic First, we have foundational logic for loading sleep study and health data files.

[7]:
```
import os
import pandas as pd


def init_study_list():
    return [x[:-4] for x in os.listdir(SLEEP_DATA_ROOT) if x.endswith('edf')]


def init_age_file():
    new_fn = 'age_file.csv'
    age_path = os.path.join(HEALTH_DATA_ROOT, 'SLEEP_STUDY.csv')

    df = pd.read_csv(age_path, sep=',', dtype='str')
    df['FILE_NAME'] = df["STUDY_PAT_ID"].str.cat(df["SLEEP_STUDY_ID"], sep='_')

    df.to_csv(new_fn, columns=["FILE_NAME", "AGE_AT_SLEEP_STUDY_DAYS"],␣
 ↪index=False)
    return os.path.abspath(new_fn)


def load_health_info(name: str, convert_datetime: bool = True):
    assert type(name) == str

    path = os.path.join(HEALTH_DATA_ROOT, name)
    df = pd.read_csv(path)

    if convert_datetime:
        if name == DEMOGRAPHIC:
            df['BIRTH_DATE'] = pd.to_datetime(df['BIRTH_DATE'],␣
 ↪infer_datetime_format=True)
        elif name == DIAGNOSIS:
            df['DX_START_DATETIME'] = pd.to_datetime(df['DX_START_DATETIME'],␣
 ↪infer_datetime_format=True)
            df['DX_END_DATETIME'] = pd.to_datetime(df['DX_END_DATETIME'],␣
 ↪infer_datetime_format=True)
        elif name == ENCOUNTER:
            df['ENCOUNTER_DATE'] = pd.to_datetime(df['ENCOUNTER_DATE'],␣
 ↪infer_datetime_format=True)
            df['VISIT_START_DATETIME'] = pd.
 ↪to_datetime(df['VISIT_START_DATETIME'], infer_datetime_format=True)
            df['VISIT_END_DATETIME'] = pd.to_datetime(df['VISIT_END_DATETIME'],␣
 ↪infer_datetime_format=True)
            df['ADT_ARRIVAL_DATETIME'] = pd.
 ↪to_datetime(df['ADT_ARRIVAL_DATETIME'], infer_datetime_format=True)
```

```
            df['ED_DEPARTURE_DATETIME'] = pd.
 ↪to_datetime(df['ED_DEPARTURE_DATETIME'], infer_datetime_format=True)
        elif name == MEASUREMENT:
            df['MEAS_RECORDED_DATETIME'] = pd.
 ↪to_datetime(df['MEAS_RECORDED_DATETIME'], infer_datetime_format=True)
        elif name == MEDICATION:
            df['MED_START_DATETIME'] = pd.to_datetime(df['MED_START_DATETIME'],␣
 ↪infer_datetime_format=True)
            df['MED_END_DATETIME'] = pd.to_datetime(df['MED_END_DATETIME'],␣
 ↪infer_datetime_format=True)
            df['MED_ORDER_DATETIME'] = pd.to_datetime(df['MED_ORDER_DATETIME'],␣
 ↪infer_datetime_format=True)
            df['MED_TAKEN_DATETIME'] = pd.to_datetime(df['MED_TAKEN_DATETIME'],␣
 ↪infer_datetime_format=True)
        elif name == PROCEDURE:
            df['PROCEDURE_DATETIME'] = pd.to_datetime(df['PROCEDURE_DATETIME'],␣
 ↪infer_datetime_format=True)
        elif name == PROCEDURE_SURG_HX:
            df['PROC_NOTED_DATE'] = pd.to_datetime(df['PROC_NOTED_DATE'],␣
 ↪infer_datetime_format=True)
            df['PROC_START_TIME'] = pd.to_datetime(df['PROC_START_TIME'],␣
 ↪infer_datetime_format=True)
            df['PROC_END_TIME'] = pd.to_datetime(df['PROC_END_TIME'],␣
 ↪infer_datetime_format=True)
        elif name == SLEEP_STUDY_NAME:
            df['SLEEP_STUDY_START_DATETIME'] = pd.
 ↪to_datetime(df['SLEEP_STUDY_START_DATETIME'])

    return df


if SHOULD_PREPROCESS:
    print(f'SHOULD_PREPROCESS is true, so loading sleep study information into␣
 ↪memory')
    ss_study_list = init_study_list()
    age_fn = init_age_file()
    SLEEP_STUDY = load_health_info("SLEEP_STUDY.csv", False)
```

Preprocessing code Next, the actual preprocessing code.

[8]:
```
import concurrent.futures
import os.path
import sys
from datetime import datetime
import pandas as pd
import mne
```

```python
import numpy as np
from biosppy.signals.ecg import hamilton_segmenter, correct_rpeaks
from biosppy.signals import tools as st
from mne import make_fixed_length_events
from scipy.interpolate import splev, splrep
from itertools import compress

mne.set_log_file('log.txt', overwrite=False)

CHUNK_DURATION = 30.0
FREQ = 64.0

POS_EVENT_DICT: dict[str, int] = {
    "Obstructive Hypopnea": 1,
    "Hypopnea": 1,
    "hypopnea": 1,
    "Mixed Hypopnea": 1,
    "Central Hypopnea": 1,

    "Obstructive Apnea": 2,
    "Central Apnea": 2,
    "Mixed Apnea": 2,
    "apnea": 2,
    "obstructive apnea": 2,
    "central apnea": 2,
    "Apnea": 2,
}

WAKE_DICT: dict[str, int] = {
    "Sleep stage W": 10
}


########################################## Annotation Modifier functions␣
 ↪##########################################
def identity(df):
    return df


def apnea2bad(df):
    df = df.replace(r'.*pnea.*', 'badevent', regex=True)
    print("bad replaced!")
    return df


def wake2bad(df):
    return df.replace("Sleep stage W", 'badevent')
```

```
def change_duration(df, label_dict=POS_EVENT_DICT, duration=CHUNK_DURATION):
    for key in label_dict:
        df.loc[df.description == key, 'duration'] = duration
    print("change duration!")
    return df


def preprocess(i, annotation_modifier, out_dir, ahi_dict):
    is_apnea_available, is_hypopnea_available = True, True
    study = ss.data.study_list[i]

    # print(f"loading study {study}")
    raw = ss.data.load_study(study, annotation_modifier, verbose=True)

    ########################################   CHECK CRITERIA FOR SS  ␣
 ↪#####################################################
    if not all([name in raw.ch_names for name in channels]):
        print("study " + str(study) + " skipped since insufficient channels",␣
 ↪file=sys.stderr)
        return 0

    ahi_value = ahi_dict.get(study, None)
    if ahi_value is None:
        print(ahi_dict)
        print("study " + str(study) + " skipped since AHI is MISSING.  Is AHI.
 ↪csv out of date?", file=sys.stderr)
        return 0

    if ahi_value < THRESHOLD:
        print("study " + str(study) + " skipped since low AHI ---  AHI = " +␣
 ↪str(ahi_value), file=sys.stderr)
        return 0

    try:
        apnea_events, event_ids = mne.events_from_annotations(
            raw,
            event_id=POS_EVENT_DICT,
            chunk_duration=1.0,
            verbose=None
        )
        # print('|')
    except ValueError:
        print("No Chunk found!", file=sys.stderr)
        return 0
    except Exception as e:
```

```
        print(e)
        return 0
    ########################################   CHECK CRITERIA FOR SS␣
↪##################################################
    print(str(i) + "---" + str(datetime.now().time().strftime("%H:%M:%S")) + '␣
↪--- Processing %d' % i)

    try:
        apnea_events, event_ids = mne.events_from_annotations(
            raw,
            event_id=APNEA_EVENT_DICT,
            chunk_duration=1.0,
            verbose=None
        )
    except ValueError:
        is_apnea_available = False

    try:
        hypopnea_events, event_ids = mne.events_from_annotations(
            raw,
            event_id=HYPOPNEA_EVENT_DICT,
            chunk_duration=1.0,
            verbose=None
        )
    except ValueError:
        is_hypopnea_available = False

    wake_events, event_ids = mne.events_from_annotations(
        raw,
        event_id=WAKE_DICT,
        chunk_duration=1.0,
        verbose=None
    )
  ␣
↪#############################################################################################
    sfreq = raw.info['sfreq']
    tmax = CHUNK_DURATION - 1. / sfreq

    raw = raw.pick_channels(channels, ordered=True)
    fixed_events = make_fixed_length_events(
        raw,
        id=0,
        duration=CHUNK_DURATION,
        overlap=0.
    )
    epochs = mne.Epochs(
        raw,
```

```
        fixed_events,
        event_id=[0],
        tmin=0,
        tmax=tmax,
        baseline=None,
        preload=True,
        proj=False,
        verbose=None
    )
    epochs.load_data()
    if sfreq != FREQ:
        epochs = epochs.resample(FREQ, npad='auto', n_jobs=4, verbose=None)
    data = epochs.get_data()

␣
↪##############################################################################################
    if is_apnea_available:
        apnea_events_set = set((apnea_events[:, 0] / sfreq).astype(int))
    if is_hypopnea_available:
        hypopnea_events_set = set((hypopnea_events[:, 0] / sfreq).astype(int))
    wake_events_set = set((wake_events[:, 0] / sfreq).astype(int))

    starts = (epochs.events[:, 0] / sfreq).astype(int)

    labels_apnea = []
    labels_hypopnea = []
    labels_wake = []
    total_apnea_event_second = 0
    total_hypopnea_event_second = 0

    for seq in range(data.shape[0]):
        epoch_set = set(range(starts[seq], starts[seq] + int(CHUNK_DURATION)))
        if is_apnea_available:
            apnea_seconds = len(apnea_events_set.intersection(epoch_set))
            total_apnea_event_second += apnea_seconds
            labels_apnea.append(apnea_seconds)
        else:
            labels_apnea.append(0)

        if is_hypopnea_available:
            hypopnea_seconds = len(hypopnea_events_set.intersection(epoch_set))
            total_hypopnea_event_second += hypopnea_seconds
            labels_hypopnea.append(hypopnea_seconds)
        else:
            labels_hypopnea.append(0)

        labels_wake.append(len(wake_events_set.intersection(epoch_set)) == 0)
```

```
    ␣
 ↪#########################################################################################
    print(study + "    HAMED    " + str(len(labels_wake) - sum(labels_wake)))
    data = data[labels_wake, :, :]
    labels_apnea = list(compress(labels_apnea, labels_wake))
    labels_hypopnea = list(compress(labels_hypopnea, labels_wake))

    out_name = study + "_" + str(total_apnea_event_second) + "_" + ␣
 ↪str(total_hypopnea_event_second)
    out_path = os.path.join(out_dir, out_name)
    # print(f"Saving {study} to {out_path}.npz")
    np.savez_compressed(out_path, data=data, labels_apnea=labels_apnea,␣
 ↪labels_hypopnea=labels_hypopnea)

    return data.shape[0]


def run_preprocess(
        ahi_path: str,
        out_folder: str,
        n_studies: int = 3984,
        n_workers: int = 3
):
    if not os.path.exists(ahi_path):
        return FileNotFoundError(f'AHI file {ahi_path} was not found!')

    ahi = pd.read_csv(ahi_path)
    # filename is <patient_id>_<study>
    filenames = ahi['PatID'].astype(str) + '_' + ahi['Study'].astype(str)
    # ahi_dict = dict(zip(ahi.Study, ahi.AHI))
    ahi_dict = dict(zip(filenames, ahi['AHI']))

    if not os.path.exists(out_folder):
        os.mkdir(out_folder)

    if n_workers < 2:
        for idx in range(n_studies):
            preprocess(
                ahi_path=idx,
                out_folder=identity,
                n_studies=out_folder,
                n_workers=ahi_dict
            )
    else:
        with concurrent.futures.ThreadPoolExecutor(
                max_workers=n_workers
        ) as executor:
```

```
            executor.map(
                preprocess,
                range(n_studies),
                [identity] * n_studies,
                [out_folder] * n_studies,
                [ahi_dict] * n_studies
            )


if SHOULD_PREPROCESS:
    print(f'SHOULD_PREPROCESS is true, so running preprocessing logic')
    run_preprocess(AHI_OUT_PATH, PREPROCESS_OUT_PATH)
```

7.5 Data Loading

After data are preprocessed, we are left with a compressed numpy array file, also called an `npz` file. At this point, we are ready to take the final step before model training - data loading.

The data loader code requires familiar collation and padding logic shown below. Notably, we make use of TensorFlow's ragged tensors to simply and easily handle the task of padding.

[9]:
```
from typing import Any
import numpy.typing as npt
import tensorflow as tf
import tensorflow.ragged


def max_dimensions(
        lst: list[Any],
        level: int = 0,
        max_dims: list[int] | None = None
) -> tuple[int, ...]:
    """
    Finds the maximum dimension for each level of a nested list structure, and␣
 ↪return
    each dimension ordered in a tuple.

    For example, given a nested list like the following:

    [[1, 2, 3], [1, 2]]

    ... return the tuple (2, 3), since the first dimension has 2 elements and␣
 ↪then
    second dimension has max(2, 3) elements.

    :param lst: The list for which to get dimensions
    :param level: INTERNAL USE ONLY (the dimension we are processing)
```

```
    :param max_dims: INTERNAL USE ONLY (the current array of maximums)
    :return: a tuple of sizes, similar to torch.Tensor.shape()
    """
    if max_dims is None:
        max_dims = []

    # Extend the max_dims list if this is the deepest level we've encountered␣
 ↪so far
    if level >= len(max_dims):
        max_dims.append(len(lst))
    else:
        max_dims[level] = max(max_dims[level], len(lst))

    for item in lst:
        if isinstance(item, list):
            # Recursively process each sublist
            max_dimensions(item, level + 1, max_dims)

    return tuple(max_dims)


def pad_lists(
        lst: list[Any],
        pad_with: int = 0
) -> npt.NDArray:
    """
    Given a ragged nested list structure `lst` (i.e. where the length
    in each dimension is not uniform), return a new list with all
    levels of the list padded to the maximal length of any list in that
    dimension. Padding elements will have the same value as given in
    `pad_with`

    For example, the return value of `pad_lists([[1], [1, 2]], 0)` will
    be `[[1, 0], [1, 2]]`

    :param lst: the list to pad, if it is ragged. if it's not, this function
        is a no-op
    :param pad_with: the value to use for padding
    """
    # max_dims[0] is the number of elements (either lists or ints) we
    # need in this dimension
    assert type(lst) is list
    return tf.ragged.constant(lst).to_tensor(pad_with).numpy()
```

```
2024-05-02 17:18:59.957425: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-05-02 17:19:00.002254: E
```

```
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-05-02 17:19:00.002322: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-05-02 17:19:00.004526: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-05-02 17:19:00.014049: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-05-02 17:19:00.016053: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-05-02 17:19:01.224490: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

Then, with padding and collating handled, data loading code is as follows:

[10]:
```python
import glob
import os
import random
import sys
from typing import Any
from urllib.request import urlretrieve

import numpy as np
import pandas as pd
from scipy.signal import resample
from biosppy.signals.ecg import hamilton_segmenter, correct_rpeaks
from biosppy.signals import tools as st
from scipy.interpolate import splev, splrep

# "EOG LOC-M2",  # 0
# "EOG ROC-M1",  # 1
# "EEG C3-M2",  # 2
# "EEG C4-M1",  # 3
# "ECG EKG2-EKG",  # 4
#
# "RESP PTAF",  # 5
# "RESP AIRFLOW",  # 6
# "RESP THORACIC",  # 7
# "RESP ABDOMINAL",  # 8
```

```
# "SPO2",  # 9
# "CAPNO",  # 10

######### ADDED IN THIS STEP #########
# RRI #11
# Ramp #12
# Demo #13


SIGS = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
s_count = len(SIGS)

THRESHOLD = 3
FREQ = 64
EPOCH_DURATION = 30
ECG_SIG = 4

AHI_PATH = AHI_OUT_PATH
OUT_PATH = PREPROCESS_OUT_PATH
PATH = os.path.join(PHYSIONET_ROOT, "nch_30x64")


def extract_rri(signal, ir, CHUNK_DURATION):
    tm = np.arange(0, CHUNK_DURATION, step=1 / float(ir))  # TIME METRIC FOR␣
 ↪INTERPOLATION

    # print('filtering', signal, FREQ)
    filtered, _, _ = st.filter_signal(signal=signal, ftype="FIR",␣
 ↪band="bandpass", order=int(0.3 * FREQ),
                                      frequency=[3, 30], sampling_rate=FREQ, )
    (rpeaks,) = hamilton_segmenter(signal=filtered, sampling_rate=FREQ)
    (rpeaks,) = correct_rpeaks(signal=filtered, rpeaks=rpeaks,␣
 ↪sampling_rate=FREQ, tol=0.05)

    if 4 < len(rpeaks) < 200:  # and np.max(signal) < 0.0015 and np.min(signal)␣
 ↪> -0.0015:
        rri_tm, rri_signal = rpeaks[1:] / float(FREQ), np.diff(rpeaks) /␣
 ↪float(FREQ)
        ampl_tm, ampl_signal = rpeaks / float(FREQ), signal[rpeaks]
        rri_interp_signal = splev(tm, splrep(rri_tm, rri_signal, k=3), ext=1)
        amp_interp_signal = splev(tm, splrep(ampl_tm, ampl_signal, k=3), ext=1)

        return np.clip(rri_interp_signal, 0, 2), np.clip(amp_interp_signal, -0.
 ↪001, 0.002)
    else:
```

```
        return np.zeros((FREQ * EPOCH_DURATION)), np.zeros((FREQ *␣
↪EPOCH_DURATION))


def load_data(path: str) -> tuple[list[Any], list[Any], list[Any]]:
    '''
    given a path to the directory of sleep studies
    (i.e. DATA_ROOT/physionet.org/nch_30x64), load and process all sleep
    studies, then return them
    '''
    # demo = pd.read_csv("../misc/result.csv") # TODO

    ahi = pd.read_csv(AHI_PATH)
    filename = ahi.PatID.astype(str) + '_' + ahi.Study.astype(str)
    ahi_dict = dict(zip(filename, ahi.AHI))
    root_dir = os.path.expanduser(path)
    file_list = os.listdir(root_dir)
    length = len(file_list)

    # print(f"Using AHI from {AHI_PATH}")
    # print(f"Using npz files from {root_dir}")
    # print(f"Files {file_list}")

    study_event_counts = {}
    apnea_event_counts = {}
    hypopnea_event_counts = {}
    ############################################ Count the respiratory events␣
↪##############################################
    for i in range(length):
        # skip directories
        if os.path.isdir(file_list[i]):
            continue

        # print(f"Processing {file_list[i]}")
        try:
            # parts = file_list[i].split("_")
            # parts[0] should be nch

            patient_id = (file_list[i].split("_")[0])
            study_id = (file_list[i].split("_")[1])
            apnea_count = int((file_list[i].split("_")[2]))
            hypopnea_count = int((file_list[i].split("_")[3]).split(".")[0])
        except Exception as e:
            print(f"Filename mismatch. Skipping {file_list[i]} ({e})", file=sys.
↪stderr)
            continue
        filename = f"{patient_id}_{study_id}"
```

```
        ahi_value = ahi_dict.get(filename, None)
        if ahi_value is None:
            print(f"Sleep study {filename} is not found in AHI.csv.  Skipping␣
↪{file_list[i]}")
            continue

        try:
            if ahi_value > THRESHOLD:
                apnea_event_counts[patient_id] = apnea_event_counts.
↪get(patient_id, 0) + apnea_count
                hypopnea_event_counts[patient_id] = hypopnea_event_counts.
↪get(patient_id, 0) + hypopnea_count
                study_event_counts[patient_id] = study_event_counts.
↪get(patient_id, 0) + apnea_count + hypopnea_count
        except Exception as e:
            print(f"File structure problem.  Skipping {file_list[i]} ({e})",␣
↪file=sys.stderr)
            continue

    apnea_event_counts = sorted(apnea_event_counts.items(), key=lambda item:␣
↪item[1])
    hypopnea_event_counts = sorted(hypopnea_event_counts.items(), key=lambda␣
↪item: item[1])
    study_event_counts = sorted(study_event_counts.items(), key=lambda item:␣
↪item[1])

    ################################### Fold the data based on number of␣
↪respiratory events ##########################
    folds = []
    for i in range(5):
        folds.append(study_event_counts[i::5])

    # print('FOLDS:', folds)

    x = []
    y_apnea = []
    y_hypopnea = []
    counter = 0
    for idx, fold in enumerate(folds):
        first = True
        aggregated_data = None
        aggregated_label_apnea = None
        aggregated_label_hypopnea = None
        for patient in fold:
            counter += 1
            # print(counter)
```

```
            glob_path = os.path.join(PATH, patient[0] + "_*")
            # print("glob path", glob_path)
            for study in glob.glob(glob_path):
                study_data = np.load(study)

                signals = study_data['data']
                labels_apnea = study_data['labels_apnea']
                labels_hypopnea = study_data['labels_hypopnea']

                identifier = study.split(os.path.sep)[-1].split('_')[0] + "_" +␣
↪study.split(os.path.sep)[-1].split('_')[
                    1]
                # print(identifier)
                # demo_arr = demo[demo['id'] == identifier].
↪drop(columns=['id']).to_numpy().squeeze() # TODO

                y_c = labels_apnea + labels_hypopnea
                neg_samples = np.where(y_c == 0)[0]
                pos_samples = list(np.where(y_c > 0)[0])
                ratio = len(pos_samples) / len(neg_samples)
                neg_survived = []
                for s in range(len(neg_samples)):
                    if random.random() < ratio:
                        neg_survived.append(neg_samples[s])
                samples = neg_survived + pos_samples
                signals = signals[samples, :, :]
                labels_apnea = labels_apnea[samples]
                labels_hypopnea = labels_hypopnea[samples]

                data = np.zeros((signals.shape[0], EPOCH_DURATION * FREQ,␣
↪s_count + 3))
                for i in range(signals.shape[0]):  # for each epoch
                    # data[i, :len(demo_arr), -1] = demo_arr TODO
                    data[i, :, -2], data[i, :, -3] = extract_rri(signals[i,␣
↪ECG_SIG, :], FREQ, float(EPOCH_DURATION))
                    for j in range(s_count):  # for each signal
                        data[i, :, j] = resample(signals[i, SIGS[j], :],␣
↪EPOCH_DURATION * FREQ)

                if first:
                    aggregated_data = data
                    aggregated_label_apnea = labels_apnea
                    aggregated_label_hypopnea = labels_hypopnea
                    first = False
                else:
                    aggregated_data = np.concatenate((aggregated_data, data),␣
↪axis=0)
```

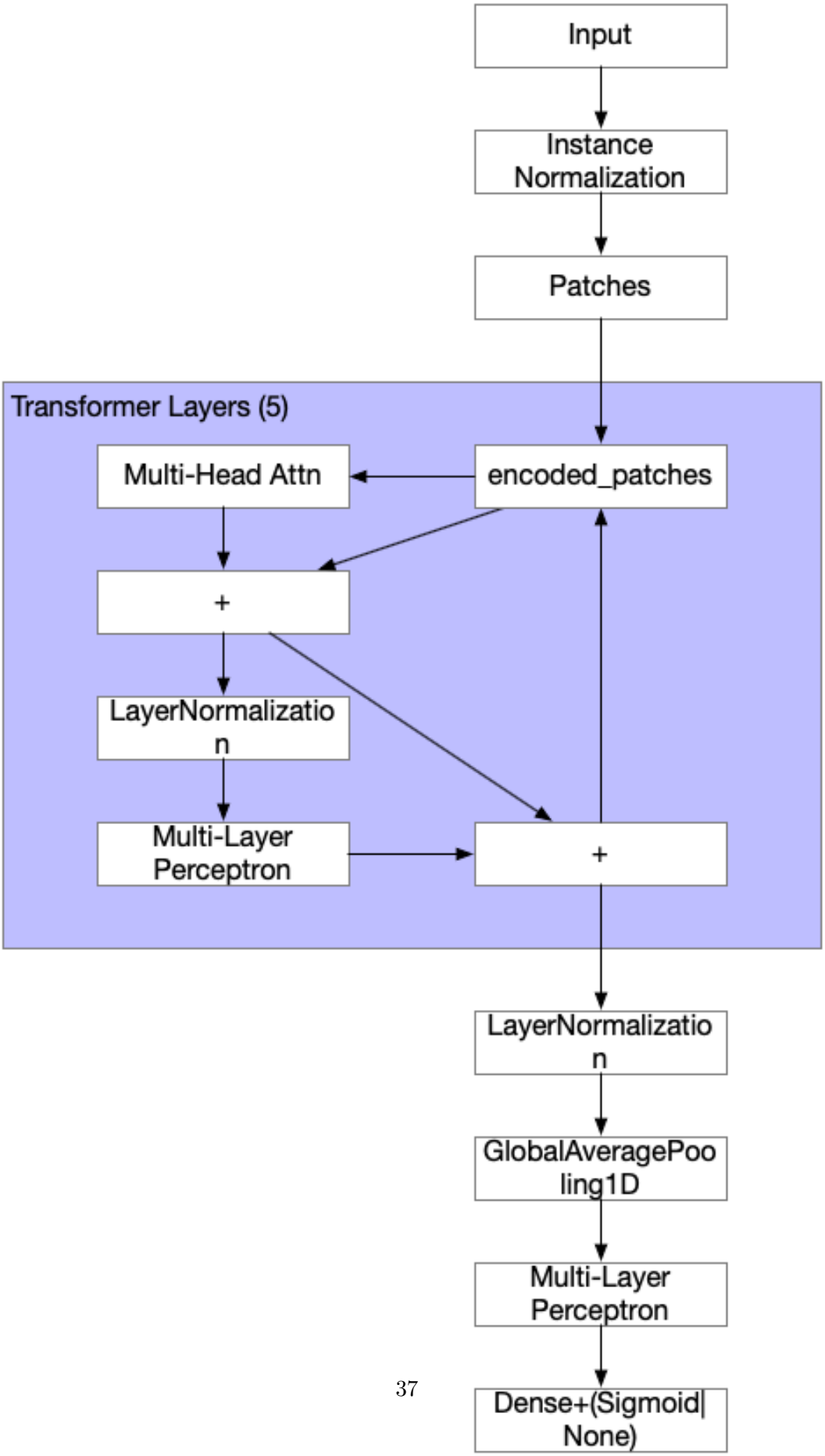
```
                    aggregated_label_apnea = np.
 ↪concatenate((aggregated_label_apnea, labels_apnea), axis=0)
                    aggregated_label_hypopnea = np.
 ↪concatenate((aggregated_label_hypopnea, labels_hypopnea), axis=0)

        if aggregated_data is not None:
            x.append(aggregated_data.tolist())
        if aggregated_label_apnea is not None:
            y_apnea.append(aggregated_label_apnea.tolist())
        if aggregated_label_hypopnea is not None:
            y_hypopnea.append(aggregated_label_hypopnea.tolist())

    return x, y_apnea, y_hypopnea


def list_lengths(lst):
    """
    Gets all the individual lengths of a list
    :param lst:
    :return:
    """
    if isinstance(lst, list):
        # For each item in the list, recursively process it if it's a list
        # Otherwise, the item itself is not counted and is represented as None␣
 ↪for non-list items
        sublengths = [list_lengths(item) for item in lst]
        if len([*filter(lambda v: v is not None, sublengths)]) == 0:
            return len(lst)
        # Instead of returning None for non-list items, you could choose to␣
 ↪omit them or handle differently
        return len(lst), sublengths  # Return the length of the current list␣
 ↪and the structure
    # Return None or some indication for non-list items, if needed
    return None


def run_load_data():
    raw_data_root = os.path.join(PHYSIONET_ROOT, "nch_30x64")
    x, y_apnea, y_hypopnea = load_data(raw_data_root)

    # these output the maximum size for dimension.
    # If we're going to make this a consistent size without truncating,
    # this is the size to make it
    print(f"Padded X.shape:{max_dimensions(x)}")
    x_norm = pad_lists(x, 0)

    print(f"Padded Y_a shape: {max_dimensions(y_apnea)}")
```

```
    y_apnea_norm = pad_lists(y_apnea, 0)

    print(f"Padded Y_h.shape:{max_dimensions(y_hypopnea)}")
    y_hypopnea_norm = pad_lists(y_hypopnea, 0)

    print(f"Saving to {OUT_PATH}")
    np.savez_compressed(
        OUT_PATH,
        x=x_norm,
        y_apnea=y_apnea_norm,
        y_hypopnea=y_hypopnea_norm,
    )


if SHOULD_PREPROCESS:
    print(f'SHOULD_PREPROCESS set to true')
    print(f'PREPROCESS_OUT_PATH={PREPROCESS_OUT_PATH}')
    run_load_data()
```

8 Model

As discussed above, the model this paper proposes is based on the transformer architecture. While this architecture has led to excellent results in NLP and other applications, it has, to our knowledge, not been studied in the context of sleep apnea. It is implemented (although broken, as we discuss elsewhere in this notebook) in the paper's accompanying open-source repository on GitHub.

Like many other transformer-based architectures, this model has several other components, illustrated below.


Input
Instance Normalization
Patches
Transformer Layers (5)
Multi-Head Attn
encoded_patches
+
LayerNormalizatio n
Multi-Layer Perceptron
+
LayerNormalizatio n
GlobalAveragePoo ling1D
Multi-Layer Perceptron
Dense+(Sigmoid| None)

As seen in this architecture, inputs, which are primarily pre-processed signals data, are normalized and passed into the transformer, which includes at least one multi-headed attention mechanism and one dense layer. Finally, the output of the transformer is passed through a normalization layer, pooling layer, fully-connected layer (labeled "Multi-layer perceptron" in the illustration) and a final activation function.

As mentioned in previous sections, we provide numerous pre-trained models because we are unable to share raw, preprocessed, or loaded data in any form due to licensing issues. Model code is shown in subsequent sections.

8.1 Preprocessed data loading

All code prior to this section relies on raw data being present. After running that code, a single `npz` (numpy compressed file, discussed above) is written to disk. Below is code to ensure that `npz` file is in place and, if not, download it if the `SHOULD_PREPROCESS` flag is set to `False`.

[11]:
```
import os


def dl_report_hook(blcks_sofar: int, blck_size: int, tot_file_size: int) ->␣
 ↪None:
    print(
        f'({blcks_sofar} blocks downloaded), '
        f'{blcks_sofar * blck_size} of {tot_file_size}'
    )


if os.path.exists(PREPROCESS_OUT_PATH):
    # if the file exists, just move on regardless of whether it was generated
    # in this notebook or downloaded
    print(f'preprocessed file found at expected location {PREPROCESS_OUT_PATH}')
elif SHOULD_PREPROCESS:
    # otherwise, the file was not found but the preprocessing flag was set,
    # so this notebook should have processed it.
    raise FileNotFoundError(
        f'ERROR: preprocessed data at path {PREPROCESS_OUT_PATH} does '
        'not exist. please check that preprocessing succeeded'
    )
else:
    print(
        f'preprocessed file {PREPROCESS_OUT_PATH} does not exist,'
        f'downloading from {PREPROCESS_URL}'
    )

    parent_dir, _ = os.path.split(PREPROCESS_OUT_PATH)
    try:
        os.makedirs(parent_dir)
    except:
```

```
        pass

    # just a simple function to report the progress of the download operation.
    # such reporting may be nice since this file could take a long time to␣
 ↪download
    # on slow connections.

    urlretrieve(
        url=PREPROCESS_URL,
        filename=PREPROCESS_OUT_PATH,
        reporthook=dl_report_hook,
    )
```

```
preprocessed file found at expected location
/tmp/data/physionet.org/nch_30x64.npz
```

8.2 Transformer layers

Since model code is extensive, we split it up into two sections. We show the transformer model code below:

[12]:
```
import keras
import tensorflow as tf
import tensorflow_addons as tfa
from keras import Model
from keras.activations import sigmoid, relu
from keras.layers import Dense, Dropout, Reshape, LayerNormalization,␣
 ↪MultiHeadAttention, Add, Flatten, Input, Layer, \
    GlobalAveragePooling1D, AveragePooling1D, Concatenate,␣
 ↪SeparableConvolution1D, Conv1D
from keras.regularizers import L2


class Patches(Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, input):
        input = input[:, tf.newaxis, :, :]
        batch_size = tf.shape(input)[0]
        patches = tf.image.extract_patches(
            images=input,
            sizes=[1, 1, self.patch_size, 1],
            strides=[1, 1, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
```

```
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches,
                             [batch_size, -1, patch_dims])
        return patches


class PatchEncoder(Layer):
    def __init__(self, num_patches, projection_dim, l2_weight):
        super(PatchEncoder, self).__init__()
        self.projection_dim = projection_dim
        self.l2_weight = l2_weight
        self.num_patches = num_patches
        self.projection = Dense(units=projection_dim,␣
 ↪kernel_regularizer=L2(l2_weight),
                                bias_regularizer=L2(l2_weight))

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch)  # + self.position_embedding(positions)
        return encoded


def mlp(x, hidden_units, dropout_rate, l2_weight):
    for _, units in enumerate(hidden_units):
        x = Dense(units, activation=None, kernel_regularizer=L2(l2_weight),␣
 ↪bias_regularizer=L2(l2_weight))(x)
        x = tf.nn.gelu(x)
        x = Dropout(dropout_rate)(x)
    return x


def create_transformer_model(input_shape, num_patches,
                             projection_dim, transformer_layers,
                             num_heads, transformer_units, mlp_head_units,
                             num_classes, drop_out, reg, l2_weight,␣
 ↪demographic=False):
    if reg:
        activation = None
    else:
        activation = 'sigmoid'
    inputs = Input(shape=input_shape)
    patch_size = input_shape[0] / num_patches
    if demographic:
        normalized_inputs = tfa.layers.InstanceNormalization(axis=-1,␣
 ↪epsilon=1e-6, center=False, scale=False,
                                                             ␣
 ↪beta_initializer="glorot_uniform",
```

```
                                                                    ␣
↪gamma_initializer="glorot_uniform")(inputs[:, :, :-1])
        demo = inputs[:, :12, -1]

    else:
        normalized_inputs = tfa.layers.InstanceNormalization(axis=-1,␣
↪epsilon=1e-6, center=False, scale=False,
                                                                    ␣
↪beta_initializer="glorot_uniform",
                                                                    ␣
↪gamma_initializer="glorot_uniform")(inputs)

    # patches = Reshape((num_patches, -1))(normalized_inputs)
    patches = Patches(patch_size=patch_size)(normalized_inputs)
    encoded_patches = PatchEncoder(num_patches=num_patches,␣
↪projection_dim=projection_dim, l2_weight=l2_weight)(patches)
    for i in range(transformer_layers):
        x1 = encoded_patches  #␣
↪LayerNormalization(epsilon=1e-6)(encoded_patches) # TODO
        attention_output = MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=drop_out,␣
↪kernel_regularizer=L2(l2_weight),  # i *
            bias_regularizer=L2(l2_weight))(x1, x1)
        x2 = Add()([attention_output, encoded_patches])
        x3 = LayerNormalization(epsilon=1e-6)(x2)
        x3 = mlp(x3, transformer_units, drop_out, l2_weight)  # i *
        encoded_patches = Add()([x3, x2])

    x = LayerNormalization(epsilon=1e-6)(encoded_patches)
    x = GlobalAveragePooling1D()(x)
    #x = Concatenate()([x, demo])
    features = mlp(x, mlp_head_units, 0.0, l2_weight)

    logits = Dense(num_classes, kernel_regularizer=L2(l2_weight),␣
↪bias_regularizer=L2(l2_weight),
                   activation=activation)(features)

    return tf.keras.Model(inputs=inputs, outputs=logits)


def create_hybrid_transformer_model(input_shape):
    transformer_units = [32, 32]
    transformer_layers = 2
    num_heads = 4
    l2_weight = 0.001
    drop_out = 0.25
```

```
  mlp_head_units = [256, 128]
  num_patches = 30
  projection_dim = 32

  # Conv1D(32...
  input1 = Input(shape=input_shape)
  conv11 = Conv1D(16, 256)(input1)  #13
  conv12 = Conv1D(16, 256)(input1)  #13
  conv13 = Conv1D(16, 256)(input1)  #13

  pwconv1 = SeparableConvolution1D(32, 1)(input1)
  pwconv2 = SeparableConvolution1D(32, 1)(pwconv1)

  conv21 = Conv1D(16, 256)(conv11)  # 7
  conv22 = Conv1D(16, 256)(conv12)  # 7
  conv23 = Conv1D(16, 256)(conv13)  # 7

  concat = keras.layers.concatenate([conv21, conv22, conv23], axis=-1)
  concat = Dense(64, activation=relu)(concat)  #192
  concat = Dense(64, activation=sigmoid)(concat)  #192
  concat = SeparableConvolution1D(32, 1)(concat)
  concat = keras.layers.concatenate([concat, pwconv2], axis=1)


␣
↪#############################################################################################
  patch_size = input_shape[0] / num_patches

  normalized_inputs = tfa.layers.InstanceNormalization(axis=-1, epsilon=1e-6,␣
↪center=False, scale=False,
                                                       ␣
↪beta_initializer="glorot_uniform",
                                                       ␣
↪gamma_initializer="glorot_uniform")(concat)

  # patches = Reshape((num_patches, -1))(normalized_inputs)
  patches = Patches(patch_size=patch_size)(normalized_inputs)
  encoded_patches = PatchEncoder(num_patches=num_patches,␣
↪projection_dim=projection_dim, l2_weight=l2_weight)(patches)
  for i in range(transformer_layers):
      x1 = encoded_patches  #␣
↪LayerNormalization(epsilon=1e-6)(encoded_patches) # TODO
      attention_output = MultiHeadAttention(
          num_heads=num_heads, key_dim=projection_dim, dropout=drop_out,␣
↪kernel_regularizer=L2(l2_weight),  # i *
          bias_regularizer=L2(l2_weight))(x1, x1)
      x2 = Add()([attention_output, encoded_patches])
```

```
        x3 = LayerNormalization(epsilon=1e-6)(x2)
        x3 = mlp(x3, transformer_units, drop_out, l2_weight)  # i *
        encoded_patches = Add()([x3, x2])

    x = LayerNormalization(epsilon=1e-6)(encoded_patches)
    x = GlobalAveragePooling1D()(x)
    #x = Concatenate()([x, demo])
    features = mlp(x, mlp_head_units, 0.0, l2_weight)

    logits = Dense(1, kernel_regularizer=L2(l2_weight),␣
↪bias_regularizer=L2(l2_weight),
                   activation='sigmoid')(features)


  ␣
↪#########################################################################################

    model = Model(inputs=input1, outputs=logits)
    return model
```

```
/root/miniconda3/envs/dlhproj/lib/python3.11/site-
packages/tensorflow_addons/utils/tfa_eol_msg.py:23: UserWarning:

TensorFlow Addons (TFA) has ended development and introduction of new features.
TFA has entered a minimal maintenance and release mode until a planned end of
life in May 2024.
Please modify downstream libraries to take dependencies from other repositories
in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).

For more information see: https://github.com/tensorflow/addons/issues/2807

  warnings.warn(
```

8.3 Supporting layers and Alternate Models

As we show above, several layers surround the transformer layers. These include several 1-D convolutions, activations, concatenations, normalizations, and fully-connected layers. This repository also contains alternate models for comparison with the Paper's Transformer model. Code for these layers is as follows:

[13]:
```
import keras
from keras import Input, Model
from keras.layers import Dense, Flatten, MaxPooling2D, Conv2D,␣
↪BatchNormalization, LSTM, Bidirectional, Permute, \
    Reshape, GRU, Conv1D, MaxPooling1D, Activation, Dropout,␣
↪GlobalAveragePooling1D, multiply, MultiHeadAttention, Add, \
    LayerNormalization, SeparableConvolution1D
from keras.models import Sequential
```

```
from keras.activations import relu, sigmoid
from keras.regularizers import l2
import tensorflow_addons as tfa


def create_cnn_model(input_shape):
    model = Sequential()
    for i in range(5):  # 10
        model.add(Conv1D(45, 32, padding='same'))
        model.add(BatchNormalization())
        model.add(Activation(relu))
        model.add(MaxPooling1D())
        model.add(Dropout(0.5))

    model.add(Flatten())
    for i in range(2):  #4
        model.add(Dense(512))
        model.add(BatchNormalization())
        model.add(Activation(relu))
        model.add(Dropout(0.5))

    model.add(Dense(1, activation='sigmoid'))

    return model


def create_cnnlstm_model(input_a_shape, weight=1e-3):
    cnn_filters = 32  # 128
    cnn_kernel_size = 4  # 4
    input1 = Input(shape=input_a_shape)
    input1 = tfa.layers.InstanceNormalization(axis=-1, epsilon=1e-6,␣
 ↪center=False, scale=False,
                                              beta_initializer="glorot_uniform",
                                              ␣
 ↪gamma_initializer="glorot_uniform")(input1)
    x1 = Conv1D(cnn_filters, cnn_kernel_size, activation='relu')(input1)
    x1 = Conv1D(cnn_filters, cnn_kernel_size, activation='relu')(x1)
    x1 = BatchNormalization()(x1)
    x1 = MaxPooling1D()(x1)

    x1 = Conv1D(cnn_filters, cnn_kernel_size, activation='relu')(x1)
    x1 = BatchNormalization()(x1)
    x1 = MaxPooling1D()(x1)

    x1 = Conv1D(cnn_filters, cnn_kernel_size, activation='relu')(x1)
    x1 = BatchNormalization()(x1)
    x1 = MaxPooling1D()(x1)
```

```
    x1 = LSTM(32, return_sequences=True)(x1)  #256
    x1 = LSTM(32, return_sequences=True)(x1)  #256
    x1 = LSTM(32)(x1)  #256
    x1 = Flatten()(x1)

    x1 = Dense(32, activation='relu')(x1)  #64
    x1 = Dense(32, activation='relu')(x1)  #64
    outputs = Dense(1, activation='sigmoid')(x1)

    model = Model(inputs=input1, outputs=outputs)
    return model


def create_semscnn_model(input_a_shape):
    input1 = Input(shape=input_a_shape)
    # input1 = tfa.layers.InstanceNormalization(axis=-1, epsilon=1e-6,␣
 ↪center=False, scale=False,
    #                                           ␣
 ↪beta_initializer="glorot_uniform",
    #                                           ␣
 ↪gamma_initializer="glorot_uniform")(input1)
    x1 = Conv1D(45, 32, strides=1)(input1)  #kernel_size=11
    x1 = Conv1D(45, 32, strides=2)(x1)  #64 kernel_size=11
    x1 = BatchNormalization()(x1)
    x1 = Activation(relu)(x1)
    x1 = MaxPooling1D()(x1)

    x1 = Conv1D(45, 32, strides=2)(x1)  #64 kernel_size=11
    x1 = BatchNormalization()(x1)
    x1 = Activation(relu)(x1)
    x1 = MaxPooling1D()(x1)

    x1 = Conv1D(45, 32, strides=2)(x1)  #64 kernel_size=11
    x1 = BatchNormalization()(x1)
    x1 = Activation(relu)(x1)
    x1 = MaxPooling1D()(x1)

    squeeze = Flatten()(x1)
    excitation = Dense(128, activation='relu')(squeeze)
    excitation = Dense(64, activation='relu')(excitation)
    logits = Dense(1, activation='sigmoid')(excitation)
    model = Model(inputs=input1, outputs=logits)
    return model


model_dict = {
```

```
    "cnn": create_cnn_model((60 * 32, 3)),
    "sem-mscnn": create_semscnn_model((60 * 32, 3)),
    "cnn-lstm": create_cnnlstm_model((60 * 32, 3)),
    "hybrid": create_hybrid_transformer_model((60 * 32, 3)),
}


def get_model(config):
    if config["model_name"].split('_')[0] == "Transformer":
        return create_transformer_model(input_shape=(60 * 32,␣
 ↪len(config["channels"])),
                                        num_patches=config["num_patches"],␣
 ↪projection_dim=config["transformer_units"],
                                        ␣
 ↪transformer_layers=config["transformer_layers"],␣
 ↪num_heads=config["num_heads"],
                                        ␣
 ↪transformer_units=[config["transformer_units"] * 2,
                                                            ␣
 ↪config["transformer_units"]],
                                        mlp_head_units=[256, 128],␣
 ↪num_classes=1, drop_out=config["drop_out_rate"],
                                        reg=config["regression"],␣
 ↪l2_weight=config["regularization_weight"])
    else:
        return model_dict.get(config["model_name"].split('_')[0])


def run_model():
    config = {
        "model_name": "hybrid",
        "regression": False,

        "transformer_layers": 4,  # best 5
        "drop_out_rate": 0.25,
        "num_patches": 20,  # best
        "transformer_units": 32,  # best 32
        "regularization_weight": 0.001,  # best 0.001
        "num_heads": 4,
        "epochs": 100,  # best
        "channels": [14, 18, 19, 20],
    }
    model = get_model(config)
    model.build(input_shape=(1, 60 * 32, 10))
    print(model.summary())
```

```
    return model
```

9 Model training

We now have preprocessed, loaded data and a model architecture, so we're ready to train a model. As discussed previously, we do not provide raw data for licensing reasons. Instead, we provide the following derivations of those raw data, on a limited basis:

- Preprocessed data, and
- Pre-trained model weights

To minimize resource requirements and runtime when running this notebook in this section, we will show training on a limited subset of the preprocessed data and limited number of epochs. We will also take steps to minimize resource requirements and runtime in the next section.

9.1 Supporting code

Prior to building our training loop, we need to write some utility code to allow us to manipulate training data as necessary.

[14]:
```
import numpy as np
import numpy.typing as npt


def _replace_final_dim(orig_x: npt.NDArray, final_dim_size: int) -> npt.NDArray:
    '''
    Given an NDArray `orig_x`, return a new NDArray of the same shape,
    except with an additional dimension filled with zeros and of size
    `final_dim_size`.
    '''
    orig_x_shape = orig_x.shape
    x_transform = np.zeros(
        (*orig_x_shape[:-1], final_dim_size)
    )
    assert x_transform.shape[:-1] == orig_x.shape[:-1]
    return x_transform


def transform_for_channels(x: npt.NDArray, channels: list[int]) -> npt.NDArray:
    '''
    Returns an NDArray whose shape is identical to that of the `x` parameter,
    except the final dimension is of size `len(channels)`

    Generally speaking, x has a shape similar to
    `(NUM_FOLDS, 530, 1920, NUM_CHANNELS)`. In this array, each fold thus
    has shape (530, 1920, NUM_CHANNELS).

    Since we're trying to only extract `num_channels` out into the last
```

```
    dimension, we need to create an NDArray whose final dimension matches
    that number of channels.

    The value returned from this function, which we'll call `x_transform`,
    is compatible with this number of channels since its last dimension is
    the number of channels we're trying to extract.
    '''

    num_channels = len(channels)
    assert (len(x.shape)) == 4
    x_transform = _replace_final_dim(
        orig_x=x,
        final_dim_size=num_channels,
    )
    return x_transform


def concat_all_folds(orig: npt.NDArray, except_fold: int) -> npt.NDArray:
    assert (except_fold) >= 0
    assert len(orig.shape) > 0
    # how do initialize our concatenated array:
    #
    # - if except_fold is 0, initialize with index 1
    # - if except_fold is 1, initialize with index 0
    #
    # in either case, we want to start at index 2 since we skip one of the
    # previous indices (0 and 1) and choose to start with the other.
    concat = orig[1] if except_fold == 0 else orig[0]
    start = 2 if except_fold == 0 else 1  # when except_fold was > 1, it would
 ↪skip orig[1]
    for i in range(start, orig.shape[0]):
        if i == except_fold:
            continue
        concat = np.concatenate((concat, orig[i]))
    return concat
```

9.2 Training loop

Next, the actual training loop.

[15]:
```
import os
from typing import Any
import keras
import keras.metrics
import numpy as np
from keras.callbacks import LearningRateScheduler, EarlyStopping
from keras.losses import BinaryCrossentropy
```

```
from sklearn.utils import shuffle

THRESHOLD = 1
FOLD = 5


def lr_schedule(epoch, lr):
    if epoch > 50 and (epoch - 1) % 5 == 0:
        lr *= 0.5
    return lr


def train(
        config,
        fold: int | None = None,
        force_retrain: bool = False
):
    print(f'training with config {config}, fold={fold}')

    data = np.load(config["data_path"], allow_pickle=True)

    x, y_apnea, y_hypopnea = data['x'], data['y_apnea'], data['y_hypopnea']
    print(
        f'x={x.shape}, y_apnea={y_apnea.shape}, y_hypopnea={y_hypopnea.shape}'
    )
    y = y_apnea + y_hypopnea
    ␣
    ↪##############################################################################################
    # Channel selection

    x_tmp = None
    for i in range(FOLD):
        x[i], y[i] = shuffle(x[i], y[i])
        x[i] = np.nan_to_num(x[i], nan=-1)
        if config["regression"]:
            y[i] = np.sqrt(y[i])
            y[i][y[i] != 0] += 2
        else:
            y[i] = np.where(y[i] >= THRESHOLD, 1, 0)

        channels = x[i][:, :, config["channels"]]  # CHANNEL SELECTION
        if x_tmp is None:
            x_tmp = np.zeros((FOLD, *channels.shape))

        x_tmp[i] = channels

    x = x_tmp
```

```
␣
↪#############################################################################################
    #
    # The original code for this is taken from the following link:
    #
    # https://github.com/healthylaife/Pediatric-Apnea-Detection/blob/
↪6dc5ec87ef17810c461d4738dd4f46240816999c/train.py#L39-L48
    #
    # I (Aaron) think that in the inner loop, they're just trying to create
    # one big NDArray with the concatenation of all the folds except for the
    # one on which they're currently on in the outer loop.
    #
    # Then, they train on the concatenated array. In other words, the outer
    # loop behaves similarly to epochs, with a small twist.
    #
    # They used to have the logic to do this inside the outer loop,
    # but I pulled it out.
    #
    # also note, the folds selection (commented below) didn't work because
    # they pass fold=0 into this function, which results in no training
    # whatsoever.
    if config["model_path"]:
        base_model_path = config["model_path"]
    else:
        model_name = get_model_name(config)
        model_dir = config["model_dir"]
        base_model_path = os.path.join(model_dir, model_name)

    folds = range(FOLD) if fold is None else [fold]
    # folds = range(FOLD) if fold is None else range(fold)
    print(f'iterating over {folds} fold(s)')
    for fold in folds:
        model_path = os.path.join(base_model_path, str(fold))

        if os.path.exists(model_path) and not force_retrain:
            print(
                f'Training fold {fold}: force_retrain==False and '
                f'{model_path} already exists, skipping.'
            )
            continue

        first = True
        x_train = None
        y_train = None
        for i in range(5):
```

```
            if i != fold:
                if first:
                    x_train = x[i]
                    y_train = y[i]
                    first = False
                else:
                    x_train = np.concatenate((x_train, x[i]))
                    y_train = np.concatenate((y_train, y[i]))

        model = get_model(config)
        if config["regression"]:
            model.compile(optimizer="adam", loss=BinaryCrossentropy())
            early_stopper = EarlyStopping(monitor='val_loss', patience=10,␣
↪restore_best_weights=True)

        else:
            model.compile(optimizer="adam", loss=BinaryCrossentropy(),
                          metrics=[keras.metrics.Precision(), keras.metrics.
↪Recall()])
            early_stopper = EarlyStopping(monitor='val_loss', patience=10,␣
↪restore_best_weights=True)
        lr_scheduler = LearningRateScheduler(lr_schedule)
        model.fit(x=x_train, y=y_train, batch_size=512,␣
↪epochs=config["epochs"], validation_split=0.1,
                  callbacks=[early_stopper, lr_scheduler])
       ␣
↪#############################################################################################
        print(f"saving model for fold {fold} to {model_path}")
        model.save(model_path)
        keras.backend.clear_session()


def train_age_seperated(config):
    data = np.load(config["data_path"], allow_pickle=True)
    x, y_apnea, y_hypopnea = data['x'], data['y_apnea'], data['y_hypopnea']
    y = y_apnea + y_hypopnea
   ␣
↪#############################################################################################
    for i in range(10):
        x[i], y[i] = shuffle(x[i], y[i])
        x[i] = np.nan_to_num(x[i], nan=-1)
        if config["regression"]:
            y[i] = np.sqrt(y[i])
            y[i][y[i] != 0] += 2
        else:
            y[i] = np.where(y[i] >= THRESHOLD, 1, 0)
```

```
        x[i] = x[i][:, :, config["channels"]]  # CHANNEL SELECTION

    ␣
↪#############################################################################################
    first = True
    for i in range(10):
        if first:
            x_train = x[i]
            y_train = y[i]
            first = False
        else:
            x_train = np.concatenate((x_train, x[i]))
            y_train = np.concatenate((y_train, y[i]))

    model = get_model(config)
    if config["regression"]:
        model.compile(optimizer="adam", loss=BinaryCrossentropy())
        early_stopper = EarlyStopping(
            monitor='val_loss',
            patience=10,
            restore_best_weights=True
        )

    else:
        model.compile(optimizer="adam", loss=BinaryCrossentropy(),
                      metrics=[keras.metrics.Precision(), keras.metrics.
↪Recall()])
        early_stopper = EarlyStopping(
            monitor='val_loss',
            patience=10,
            restore_best_weights=True
        )
    lr_scheduler = LearningRateScheduler(lr_schedule)
    model.fit(
        x=x_train,
        y=y_train,
        batch_size=512,
        epochs=config["epochs"],
        validation_split=0.1,
        callbacks=[early_stopper, lr_scheduler]
    )
    ␣
↪#############################################################################################
    model.save(config["model_path"] + str(0))
    keras.backend.clear_session()
```

```
# the original repository indicated that the best value for this constant
# is 200. we are setting it to a much lower number so we can illustrate
# training working without requiring a large amount of computing power
# or running for an excessively long time.
#
# feel free to increase this value, but be mindful that if you do, we make
# no guarantees about required resources or expected runtime.

def build_configs() -> dict[str, dict[str, Any]]:
    sig_dict = {"EOG": [0, 1],
                "EEG": [2, 3],
                "RESP": [5, 6],
                "SPO2": [9],
                "CO2": [10],
                "ECG": [11, 12],
                "DEMO": [13],
                }

    channel_list = [
        ["ECG", "SPO2"],
    ]
    configs: dict[str, dict[str, Any]] = {}
    for ch in channel_list:
        chs = []
        chstr = ""
        for name in ch:
            chstr += name
            chs = chs + sig_dict[name]
        if chstr in configs:
            raise ValueError(
                f"tried to create a config for channel {chstr}, "
                "but one already existed"
            )
        configs[chstr] = {
            "data_path": PREPROCESS_OUT_PATH,
            "model_path": MODEL_OUT_PATH,
            "model_name": MODEL + "_" + chstr,
            "regression": False,

            "transformer_layers": 5,  # best 5
            "drop_out_rate": 0.25,  # best 0.25
            "num_patches": 30,  # best 30 TBD
            "transformer_units": 32,  # best 32
            "regularization_weight": 0.001,  # best 0.001
            "num_heads": 4,
            "epochs": NUM_EPOCHS,
```

```
            "channels": chs,
        }
    return configs


def run_training():
    configs = build_configs()

    for chstr, config in configs.items():
        print(
            f"----------\n"
            f"training channel {chstr}..."
        )
        train(config=config, fold=None, force_retrain=True)
        print("done.")


if SKIP_TRAINING:
    print("SKIP_TRAINING is set to True, so not running training loop")
else:
    print('running training...')
    run_training()
    print("done.")
```

```
running training…
----------
training channel ECGSPO2…
training with config {'data_path': '/tmp/data/physionet.org/nch_30x64.npz',
'model_path': 'original/weights/Transformer_NOTEBOOK_2', 'model_name':
'Transformer_ECGSPO2', 'regression': False, 'transformer_layers': 5,
'drop_out_rate': 0.25, 'num_patches': 30, 'transformer_units': 32,
'regularization_weight': 0.001, 'num_heads': 4, 'epochs': 2, 'channels': [11,
12, 9]}, fold=None
x=(5, 4365, 1920, 14), y_apnea=(5, 4365), y_hypopnea=(5, 4365)
iterating over range(0, 5) fold(s)
Epoch 1/2
31/31 [==============================] - 31s 607ms/step - loss: 1.3229 -
precision: 0.6621 - recall: 0.7632 - val_loss: 1.2342 - val_precision: 0.6541 -
val_recall: 0.9582 - lr: 0.0010
Epoch 2/2
31/31 [==============================] - 16s 530ms/step - loss: 1.0241 -
precision: 0.7185 - recall: 0.8769 - val_loss: 1.0944 - val_precision: 0.6625 -
val_recall: 0.9639 - lr: 0.0010
saving model for fold 0 to original/weights/Transformer_NOTEBOOK_2/0
INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/0/assets

INFO:tensorflow:Assets written to:
```

```
original/weights/Transformer_NOTEBOOK_2/0/assets

Epoch 1/2
31/31 [==============================] - 31s 577ms/step - loss: 1.3336 -
precision: 0.6469 - recall: 0.7632 - val_loss: 1.2423 - val_precision: 0.6578 -
val_recall: 0.9481 - lr: 0.0010
Epoch 2/2
31/31 [==============================] - 15s 487ms/step - loss: 1.0459 -
precision: 0.6985 - recall: 0.8890 - val_loss: 1.0619 - val_precision: 0.6866 -
val_recall: 0.9345 - lr: 0.0010
saving model for fold 1 to original/weights/Transformer_NOTEBOOK_2/1
INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/1/assets

INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/1/assets

Epoch 1/2
31/31 [==============================] - 32s 600ms/step - loss: 1.3398 -
precision: 0.6535 - recall: 0.8192 - val_loss: 1.2708 - val_precision: 0.6447 -
val_recall: 0.9707 - lr: 0.0010
Epoch 2/2
31/31 [==============================] - 17s 556ms/step - loss: 1.0737 -
precision: 0.6848 - recall: 0.8956 - val_loss: 1.0576 - val_precision: 0.7103 -
val_recall: 0.8995 - lr: 0.0010
saving model for fold 2 to original/weights/Transformer_NOTEBOOK_2/2
INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/2/assets

INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/2/assets

Epoch 1/2
31/31 [==============================] - 29s 484ms/step - loss: 1.2925 -
precision: 0.6621 - recall: 0.7592 - val_loss: 1.2305 - val_precision: 0.6607 -
val_recall: 0.9628 - lr: 0.0010
Epoch 2/2
31/31 [==============================] - 16s 516ms/step - loss: 1.0037 -
precision: 0.7015 - recall: 0.8853 - val_loss: 1.0613 - val_precision: 0.6798 -
val_recall: 0.9176 - lr: 0.0010
saving model for fold 3 to original/weights/Transformer_NOTEBOOK_2/3
INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/3/assets

INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/3/assets

Epoch 1/2
31/31 [==============================] - 31s 515ms/step - loss: 1.2823 -
precision: 0.6699 - recall: 0.8027 - val_loss: 1.1749 - val_precision: 0.6667 -
val_recall: 0.9513 - lr: 0.0010
```

```
Epoch 2/2
31/31 [==============================] - 16s 533ms/step - loss: 0.9993 -
precision: 0.7160 - recall: 0.8808 - val_loss: 0.9746 - val_precision: 0.6998 -
val_recall: 0.9385 - lr: 0.0010
saving model for fold 4 to original/weights/Transformer_NOTEBOOK_2/4
INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/4/assets

INFO:tensorflow:Assets written to:
original/weights/Transformer_NOTEBOOK_2/4/assets

done.
done.
```

9.3 Hyperparameters

In our model training loop, the number of epochs was set at 200. A Keras `EarlyStopping` callback was used to bailout from training if the value loss does not improve for 10 epochs. Thus, while the *maximum* epochs were 200, the *actual* epochs used to train a model were typically around 70.

The starting learning rate is `0.0010`, and learning rates on and after epoch 52 were reduced using a Keras `LearningRateScheduler`. At epoch 52 and every 5 epochs thereafter, the learning rate is halved. Thus, the learning rate schedule looks like the following after the first 3 adjustments:

Epoch range	Learning Rate
1-51	0.0010
52-56	0.0005
57-62	0.00025

The subsequent adjustments continue in this pattern as expected. This learning rate decay, along with the `EarlyStopping` bailout, undoubtedly contributed to the limited number of epochs actually used. It is possible that using a larger subset of the dataset, or the whole dataset, would result in more epochs being used before the bailout is reached.

9.3.1 Other Hyperparameters Used

In addition to the above hyperparameter settings, we also did training+testing runs with the following epoch combinations:

> In all cases, similar `EarlyStopping` and `LearningRateScheduler` mechanisms were used as above.

Num. Epochs	Notes
2	Primarily used for local testing on limited computational resources
50	Done on an 8-core CPU on a DigitalOcean Droplet VM

Num. Epochs	Notes
100	Performed both on an 8-core CPU and on an NVIDIA H-100 GPU, both cases on virtualized cloud systems

9.4 Computational Requirements

Model checkpoint generation was performed on several hardware configurations, including the following:

9.4.1 Windows 11 PC

In this configuration, training was done on Ubuntu running inside a Windows Subsystem of Linux virtual machine (VM). This setup had the following hardware specifications:

- Hardware CPU: AMD Ryzen 7 3800X
  - 8 physical cores, up to 4.2 GHz
  - 16 virtual processors
- Hardware GPU: NVIDIA RTX 4070Ti
  - 12GB of GPU RAM
  - Up to 44GB shared RAM
- Hardware RAM: 64GB
- Storage: 4TB SSD

9.4.2 DigitalOcean VM

In this configuration, training was done on Ubuntu running on a DigitalOcean cloud VM (called a "droplet"). Since this is a cloud-based VM, there are relatively few guarantees about the underlying hardware. This setup had the following known specification:

- Virtual CPU: 8-core generic AMD
- Virtual GPU: None
- RAM: 32GB
- Storage: 400GB SSD

9.4.3 Paperspace VM

In this configuration, training was done on Ubuntu running on a Paperspace AI/ML-optimized cloud VM. Since this is a cloud-based VM, there are relatively few guarantees made about the underlying hardware, similar to the DigitalOcean case above. This setup had the following known specification:

- Virtual CPU: 8-core generic
- Virtual GPU: NVIDIA P4000
  - 8GB GPU RAM
- RAM: 30GB
- Storage: 1TB SSD

9.4.4 Overview of Computations

Generally speaking, computations were performed on a 500GB subset of the NCH dataset. This consumed the nearly entire system RAM in all the above hardware configurations during preprocessing. Similarly, in the GPU-enabled setups above, all the GPU-dedicated RAM was used during model training.

In the Windows setup, each epoch of each fold lasted approximately 2 seconds. The total combined training and testing time per model, including data loading and model saving was approximately 14 minutes. For the total 63 ablations computed that equates to approximately 882 minutes or 14.7 hours of total compute time, most of which was nearly 100% GPU-bound.

In the DigitalOcean setup described above, training and testing took significantly longer, and in the Paperspace setup, it took moderately longer, although we did not do an apples-to-apples comparison. Further, since the latter two configurations were cloud-based systems, we're unable to guarantee exclusive access to the underlying hardware and thus unable to guarantee a stable experiment in those cases.

10 Results and evaluations

In a traditional machine learning pipeline, we would train a model and then immediately evaluate it on a test data set. As with the last section, however, we take steps to minimize resource requirements and runtime. To that end, we have run training on the full dataset and saved the resulting model's weights (which you can find in our repository's `original/weights` directory).

In this section, however, we've built the option to instantiate our model from those weights, then evaluate it. The results of this evaluation determine how well we were able to reproduce the results claimed in the original paper.

The code shown in this section tests our preloaded model. As with the previous section, testing and evaluation code is extensive, so we split it into two separate sections.

10.1 Metrics reporting and calculation

Below, we have utility code for metrics collection and reporting. This code will be used by subsequent evaluation code.

[16]:
```
import os
import tensorflow as tf
import tensorflow_addons as tfa
import numpy as np
from sklearn.metrics import confusion_matrix, average_precision_score,␣
 ↪roc_auc_score, accuracy_score, \
   precision_recall_fscore_support


class FromLogitsMixin:
    def __init__(self, from_logits=False, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.from_logits = from_logits
```

```
    def update_state(self, y_true, y_pred, sample_weight=None):
        if self.from_logits:
            y_pred = tf.nn.sigmoid(y_pred)
        return super().update_state(y_true, y_pred, sample_weight)


class AUC(FromLogitsMixin, tf.metrics.AUC):
    ...


class BinaryAccuracy(FromLogitsMixin, tf.metrics.BinaryAccuracy):
    ...


class TruePositives(FromLogitsMixin, tf.metrics.TruePositives):
    ...


class FalsePositives(FromLogitsMixin, tf.metrics.FalsePositives):
    ...


class TrueNegatives(FromLogitsMixin, tf.metrics.TrueNegatives):
    ...


class FalseNegatives(FromLogitsMixin, tf.metrics.FalseNegatives):
    ...


class Precision(FromLogitsMixin, tf.metrics.Precision):
    ...


class Recall(FromLogitsMixin, tf.metrics.Recall):
    ...


class F1Score(FromLogitsMixin, tfa.metrics.F1Score):
    ...


class Result:
    def __init__(self):
        self.accuracy_list = []
        self.sensitivity_list = []
```

```
        self.specificity_list = []
        self.f1_list = []
        self.auroc_list = []
        self.auprc_list = []
        self.precision_list = []

    def add(self, y_test, y_predict, y_score):
        tn, fp, fn, tp = confusion_matrix(y_test, y_predict).ravel()

        acc = accuracy_score(y_test, y_predict)
        sp = 1. * tn / (tn + fp) if (tn + fp) != 0 else 0
        pr, sn, f1, _ = precision_recall_fscore_support(y_test, y_predict,␣
↪zero_division=0.0)
        auc = roc_auc_score(y_test, y_score)
        auprc = average_precision_score(y_test, y_score)

        self.accuracy_list.append(acc * 100)
        self.precision_list.append(pr * 100)
        self.sensitivity_list.append(sn * 100)
        self.specificity_list.append(sp * 100)
        self.f1_list.append(f1 * 100)
        self.auroc_list.append(auc * 100)
        self.auprc_list.append(auprc * 100)

    def get(self):
        out_str =␣
↪"================================================================================␣
↪\n"
        out_str += str(self.accuracy_list) + " \n"
        out_str += str(self.precision_list) + " \n"
        out_str += str(self.sensitivity_list) + " \n"
        out_str += str(self.specificity_list) + " \n"
        out_str += str(self.f1_list) + " \n"
        out_str += str(self.auroc_list) + " \n"
        out_str += str(self.auprc_list) + " \n"
        out_str += str("Accuracy: %.2f -+ %.3f" % (np.mean(self.accuracy_list),␣
↪np.std(self.accuracy_list))) + " \n"
        out_str += str("Precision: %.2f -+ %.3f" % (np.mean(self.
↪precision_list), np.std(self.precision_list))) + " \n"
        out_str += str(
            "Recall: %.2f -+ %.3f" % (np.mean(self.sensitivity_list), np.
↪std(self.sensitivity_list))) + " \n"
        out_str += str(
            "Specifity: %.2f -+ %.3f" % (np.mean(self.specificity_list), np.
↪std(self.specificity_list))) + " \n"
```

```
        out_str += str("F1: %.2f -+ %.3f" % (np.mean(self.f1_list), np.std(self.
↪f1_list))) + " \n"
        out_str += str("AUROC: %.2f -+ %.3f" % (np.mean(self.auroc_list), np.
↪std(self.auroc_list))) + " \n"
        out_str += str("AUPRC: %.2f -+ %.3f" % (np.mean(self.auprc_list), np.
↪std(self.auprc_list))) + " \n"

        out_str += str("$ %.1f \pm %.1f$" % (np.mean(self.accuracy_list), np.
↪std(self.accuracy_list))) + "& "
        out_str += str("$%.1f \pm %.1f$" % (np.mean(self.precision_list), np.
↪std(self.precision_list))) + "& "
        out_str += str("$%.1f \pm %.1f$" % (np.mean(self.sensitivity_list), np.
↪std(self.sensitivity_list))) + "& "
        out_str += str("$%.1f \pm %.1f$" % (np.mean(self.f1_list), np.std(self.
↪f1_list))) + "& "
        out_str += str("$%.1f \pm %.1f$" % (np.mean(self.auroc_list), np.
↪std(self.auroc_list))) + "& "

        return out_str

    def print(self):
        print(self.get())

    def save(self, path, config):
        enclosing_dir = os.path.dirname(path)
        if not os.path.exists(enclosing_dir):
            print(
                f'directory {enclosing_dir} did not exist, creating it for '
                'result saving'
            )
            os.makedirs(enclosing_dir)

        file = open(path, "w+")
        file.write(str(config))
        file.write("\n")
        file.write(self.get())
        file.flush()
        file.close()
```

10.2 Using the pretrained model

The `TEST_FROM_CHECKPOINT` flag indicates whether we should test from a model loaded from pre-trained weights (`True`), or the model we just tested (`False`). If that flag is set to `True`, we expect a zip archive of the weights to be downloadable from the value in `CHECKPOINT_URL`. The code below checks for the flag and downloads weights if necessary.

The default configuration from the beginning of this notebook is set up to download a pre-trained

model rather than using the model trained in this notebook.

[17]:
```
import os
from urllib.request import urlretrieve
import zipfile
import random
from shutil import rmtree, move


def mktmp(length: int = 20, base_dir: str = os.path.join('/', 'tmp')) -> str:
    """
    Generate a random, unique file or directory name in base_dir.  This DOES␣
 ↪NOT CREATE a file or directory, but simply returns one that doesn't exist.
    :param length:int The length of the random file/directory name.  DEFAULT 20
    :param base_dir:str A directory DEFAULT: '/tmp'
    :return: str
    """
    char_choices = [chr(v) for v in list(range(ord('0'), ord('9') + 1)) +␣
 ↪list(range(ord('a'), ord('z') + 1)) + list(
        range(ord('A'), ord('Z') + 1))]
    characters = random.choices(char_choices, k=length)
    path = os.path.join(base_dir, ''.join(characters))
    return path if not os.path.exists(path) else mktmp(length=length,␣
 ↪base_dir=base_dir)


if TEST_FROM_CHECKPOINT:
    print(
        f'TEST_FROM_CHECKPOINT is True, downloading {CHECKPOINT_URL} '
        f'and saving to {MODEL_OUT_PATH}'
    )
    zip_path = "./model.zip"

    # only download the model if the model directory doesn't exist already

    # first download the zip to the current working dir
    urlretrieve(CHECKPOINT_URL, zip_path)  #, reporthook=dl_report_hook,)

    # crate a temporary directory to work in
    tmp_dir = mktmp(base_dir=MODEL_DIR)
    os.makedirs(tmp_dir)

    # extract into the temp directory
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        print(f'Extracting pretrained model to {tmp_dir}')
        zip_ref.extractall(tmp_dir)
```

```
    # rename the downloaded model to match the expected directory name
    _, expected_model_name = os.path.split(MODEL_OUT_PATH)
    move(os.path.join(tmp_dir, os.listdir(tmp_dir)[0]), os.path.join(tmp_dir,␣
 ↪expected_model_name))
    tmp_model_path = os.path.join(tmp_dir, expected_model_name)

    if os.path.exists(MODEL_OUT_PATH):
        print(f'Removing existing model to use pretrained model')
        rmtree(MODEL_OUT_PATH)

    # move the pre-trained model into place
    move(tmp_model_path, MODEL_DIR)

    # cleanup
    rmtree(tmp_dir)
    os.remove(zip_path)

else:
    if not os.path.exists(MODEL_OUT_PATH):
        raise FileNotFoundError(
            f"TEST_FROM_CHECKPOINT is False, but {MODEL_OUT_PATH} was not "
            f"found. please make sure model training succeeded"
        )
```

```
TEST_FROM_CHECKPOINT is True, downloading
https://dlhproject.sfo3.cdn.digitaloceanspaces.com/Transformer_SPO2ECG_200.zip
and saving to original/weights/Transformer_NOTEBOOK_2
Extracting pretrained model to original/weights/eRf3hRjeSguGZkm8WFjm
Removing existing model to use pretrained model
```

10.3 Metrics calculation

The below code evaluates the model and reports it using the above reporting code.

[18]:
```
import os
from datetime import datetime

import numpy as np
from sklearn.utils import shuffle
import tensorflow as tf


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


THRESHOLD = 1
FOLD = 5
```

```
def add_noise_to_data(a):
    """
    Stand-in function for adding noise to data.

    This function is used in the codebase, hidden behind a
    config flag, and not defined anywhere in the original research
    repository, so we have defined it here as the identity function,
    just so things run properly.
    """
    print("WARNING: 'test_noise_snr' config option is a no-op")
    return a


def get_model_name(config: dict) -> str:
    """
    A simple stand-in function for getting the model name.  We use a more␣
 ↪robust system in the main repo
    :param config:dict
    :return:
    """
    name = config.get('model_name', "Unknown_")
    return name.split('_')[0]


def test(config: dict[str, str], fold=None):
    data_path = config['data_path']
    print(f'testing from {data_path}')
    data = np.load(config["data_path"], allow_pickle=True)
    ##############################################################################
    x, y_apnea, y_hypopnea = data['x'], data['y_apnea'], data['y_hypopnea']
    y = y_apnea + y_hypopnea
    x_tmp = None

    for i in range(FOLD):
        x[i], y[i] = shuffle(x[i], y[i])
        x[i] = np.nan_to_num(x[i], nan=-1)
        y[i] = np.where(y[i] >= THRESHOLD, 1, 0)

        channels = x[i][:, :, config["channels"]]
        if x_tmp is None:
            x_tmp = np.zeros((FOLD, *channels.shape))

        x_tmp[i] = channels

    x = x_tmp
```

```
    ##############################################################################
    result = Result()
    folds = range(FOLD) if fold is None else [fold]

    model_name = get_model_name(config)
    if config["model_path"]:
        model_path = config["model_path"]
    else:
        model_path = os.path.join(config["model_dir"], model_name)
    for fold in folds:
        x_test = x[fold]
        if "test_noise_snr" in config and config["test_noise_snr"]:
            x_test = add_noise_to_data(x_test, config["test_noise_snr"])

        y_test = y[
            fold
        ]  # For MultiClass keras.utils.to_categorical(y[fold], num_classes=2)
        model = tf.keras.models.load_model(os.path.join(model_path, str(fold)),␣
↪compile=False)

        predict = model.predict(x_test)
        y_score = predict
        y_predict = np.where(
            predict > 0.5, 1, 0
        )  # For MultiClass np.argmax(y_score, axis=-1)

        result.add(y_test, y_predict, y_score)

    print(
        '\n----------\n'
        'results:\n'
    )
    result.print()
    # model_name = config["model_name"]
    timestamp = datetime.now().strftime("%Y%m%d-%H%M")  # Format the date and␣
↪time as a string "YYYYMMDD-HH:mm"
    results_file = os.path.join('results', f"{model_name}-{timestamp}.txt")
    print(
        f'done, saving to {results_file}\n'
        '----------\n'
    )

    result.save(path=results_file, config=config)

    del data, x_test, y_test, model, predict, y_score, y_predict
```

```
def test_age_seperated(config):
    x = []
    y_apnea = []
    y_hypopnea = []
    for i in range(10):
        data = np.load(config["data_path"] + str(i) + ".npz", allow_pickle=True)
        x.append(data["x"])
        y_apnea.append(data["y_apnea"])
        y_hypopnea.append(data["y_hypopnea"])
    ###############################################################################
    y = np.array(y_apnea) + np.array(y_hypopnea)
    for i in range(10):
        x[i], y[i] = shuffle(x[i], y[i])
        x[i] = np.nan_to_num(x[i], nan=-1)
        y[i] = np.where(y[i] >= THRESHOLD, 1, 0)
        x[i] = x[i][:, :, config["channels"]]
    ###############################################################################
    result = Result()

    for fold in range(10):
        x_test = x[fold]
        if config.get("test_noise_snr"):
            x_test = add_noise_to_data(x_test, config["test_noise_snr"])

        y_test = y[
            fold
        ]  # For MultiClass keras.utils.to_categorical(y[fold], num_classes=2)

        # model = tf.keras.models.load_model(config["model_path"] + str(0),␣
↪compile=False)
        model = tf.keras.models.load_model(MODEL_OUT_PATH, compile=False)

        predict = model.predict(x_test)
        y_score = predict
        y_predict = np.where(
            predict > 0.5, 1, 0
        )  # For MultiClass np.argmax(y_score, axis=-1)

        result.add(y_test, y_predict, y_score)

    result.print()
    save_file = os.path.join('results', config["model_name"], ".txt")
    result.save(save_file, config)
    print(f"results saved to {save_file}")

    del data, x_test, y_test, model, predict, y_score, y_predict
```

```
configs = build_configs()
for chstr, config in configs.items():
    print(f'testing on channel {chstr}')
    test(config)
```

```
testing on channel ECGSPO2
testing from /tmp/data/physionet.org/nch_30x64.npz

WARNING:root:The given value for groups will be overwritten.

137/137 [==============================] - 4s 23ms/step

WARNING:root:The given value for groups will be overwritten.

137/137 [==============================] - 4s 24ms/step

WARNING:root:The given value for groups will be overwritten.

137/137 [==============================] - 4s 21ms/step

WARNING:root:The given value for groups will be overwritten.

137/137 [==============================] - 4s 24ms/step

WARNING:root:The given value for groups will be overwritten.

137/137 [==============================] - 4s 23ms/step

----------
results:

================================================================================
[86.3917525773196, 87.7434135166094, 88.08705612829324, 84.28407789232531,
80.20618556701031]
[array([90.82568807, 79.02439024]), array([91.51278409, 80.89089735]),
array([90.45226131, 83.91386954]), array([87.05832269, 81.0515873 ]),
array([84.61134454, 76.79804957])]
[array([87.79709117, 83.82923674]), array([89.69718065, 83.98123324]),
array([90.84354722, 83.28095537]), array([84.26040379, 84.31372549]),
array([73.83134739, 86.57810353])]
[87.79709116708052, 89.6971806474069, 90.84354722422493, 84.26040379068809,
73.8313473877177]
[array([89.28571429, 81.3559322 ]), array([90.5958868 , 82.40710293]),
array([90.64748201, 83.59621451]), array([85.63651591, 82.65048053]),
array([78.85462555, 81.39534884])]
[93.82911283487076, 94.82775755415352, 94.8880395628438, 91.73904853507773,
88.23850493753709]
[87.24682466964501, 88.19214307557768, 89.69552187333034, 87.04942894091803,
85.90361306797132]
Accuracy: 85.34 -+ 2.895
Precision: 84.61 -+ 4.956
```

```
Recall: 84.84 -+ 4.443
Specifity: 85.29 -+ 6.147
F1: 84.64 -+ 3.987
AUROC: 92.70 -+ 2.507
AUPRC: 87.62 -+ 1.269
$ 85.3 \pm 2.9$& $84.6 \pm 5.0$& $84.8 \pm 4.4$& $84.6 \pm 4.0$& $92.7 \pm 2.5$&
done, saving to results/Transformer-20240502-1725.txt
----------
```

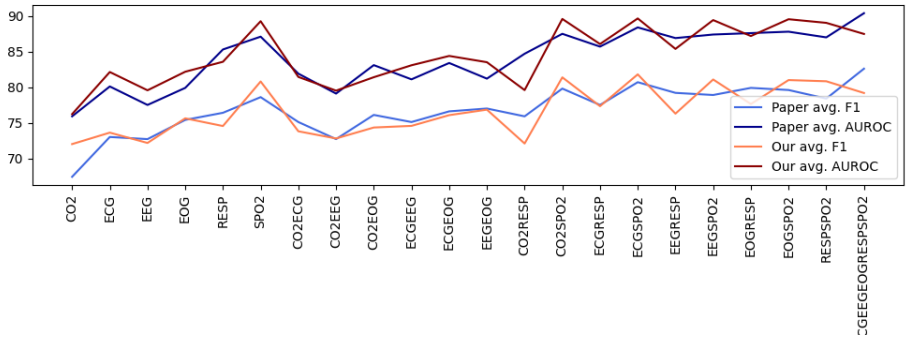
After evaluation, the output of each metric will be given in the form `<Statistic>: <mean> -+ <std dev>`. Thus, `F1: 77.31 -+ 2.880` is an F1 score (on a 0 to 100 range) with a mean across all folds of 77.31 and a standard deviation of 2.88.

10.4 Model comparison

10.4.1 Table of Results

CO2	ECG	EEG	EOG	RESP	SPO2	Mean F1	StdDev F1	Mean AU-ROC	StdDev AU-ROC
X						72.00	4.922	76.21	5.277
	X					73.61	2.884	82.13	2.299
		X				72.16	2.844	79.56	2.664
			X			75.62	2.737	82.17	2.165
				X		74.54	4.417	83.57	3.399
					X	80.80	3.011	89.27	2.300
X	X					73.80	4.433	81.44	4.658
X		X				72.79	3.511	79.51	3.396
	X	X				74.56	2.322	83.08	1.852
X			X			74.33	3.536	81.40	3.701
	X		X			76.07	2.281	84.39	1.976
		X	X			76.84	2.440	83.51	2.162
		X		X		76.28	4.883	85.38	2.884
		X			X	81.08	2.699	89.43	2.242
			X	X		77.63	4.331	87.19	1.894
			X		X	81.00	2.347	89.55	1.932
X				X		72.08	5.568	79.59	5.236
	X			X		77.33	3.228	86.06	2.001
X					X	81.38	2.867	89.59	2.099
	X				X	81.80	2.435	89.66	1.994
				X	X	80.83	2.626	89.04	2.036
X	X	X	X	X	X	79.18	3.553	87.48	3.070

These results, while not a decimal-for-decimal duplication are, overall, very close to the paper's results and particularly consistent with the trends shown in the original paper, as demonstrated by the following comparison graph.



> Code to generate this the the below data visualizations is elided from this notebook for clarity and brevity purposes, and can be found in our repository at `original/eval/plot.py`. We have provided rendered graph images instead.

In this figure, our results are shown in shades of orange with the original paper's results shown in shades of blue.

The original paper's results are more consistent across the various combinations of signals, and while the mean F1 score and the mean AUROC score are close, our results differ more substantially in the standard deviation of each value.

Some key differences account for the differences in the final calculated statistics. Chiefly among these, the size of the data set used for training differed substantially. The original paper used the entire NCH dataset, but due to bandwidth limitations on physionet.org (the repository for these data), we were only able to use a roughly 500GB subset of the data (around 25%), which was selected by the first files the `wget` command (the tool used to download these data) fetched.

Thus, we do not truly know if this was a random sample, a chronological sample, or if our dataset was skewed by some underlying sampling bias inadvertently introduced by the wget command. The quality of the results overall seems to suggest it was random, but without much more analysis of the included data against the broader dataset, that question remains open.

It is logical to conclude that a larger overall training set would make each fold proportionally larger and, thus, lead to more consistent results across each combination of folds and, in turn, lead to a smaller standard deviation for F1 and AUROC scores. Thus, we conclude the majority of the difference between the original paper's results and our results is in the dataset size used for training.

10.5 Results Beyond the Original Paper

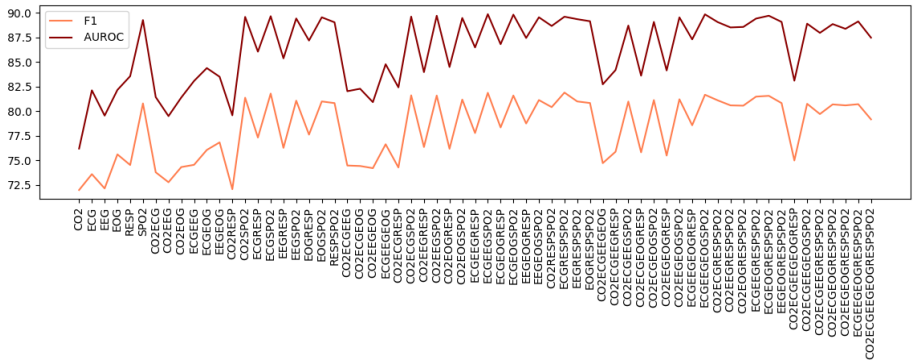
10.5.1 More Ablations

We undertook to complete all possible ablations, including those not present in the paper. The paper included combinations of one signal, combinations of two signals, and all six signals (22 in total). We ran additional ablations on combinations of three, four, and five signals (41 additional; 63 in total). Our results are shown in the table as follows:

CO2	ECG	EEG	EOG	RESP	SPO2	Mean F1	StdDev F1	Mean AU-ROC	StdDev AU-ROC
X	X	X				74.49	3.478	82.04	4.030
X	X		X			74.44	3.343	82.29	3.989
X		X	X			74.22	3.359	80.93	3.569
	X	X	X			76.65	2.167	84.78	1.793
X		X		X		76.37	4.227	83.98	3.874
	X	X		X		77.79	2.897	86.49	1.804
X		X			X	81.60	2.916	89.71	2.094
	X	X			X	81.88	2.279	89.87	1.831
		X	X	X		78.76	2.905	87.45	1.610
		X	X		X	81.14	2.704	89.55	1.911
X			X	X		76.19	3.918	84.50	2.507
	X		X	X		78.36	2.863	86.82	1.608
X			X		X	81.19	3.142	89.48	2.236
	X		X		X	81.60	2.146	89.82	1.627
X	X			X		74.29	4.232	82.43	3.876
X	X				X	81.62	3.072	89.62	2.180
		X		X	X	81.00	2.366	89.36	1.729
			X	X	X	80.84	2.865	89.15	1.924
X				X	X	80.42	3.499	88.67	2.585
	X			X	X	81.90	2.420	89.61	1.869
X	X	X	X			74.73	3.362	82.74	3.694
X	X	X		X		75.89	3.345	84.19	2.690
X	X	X			X	80.99	2.992	88.71	2.356
X		X	X	X		75.50	3.927	84.15	2.685
	X	X	X	X		78.58	2.619	87.31	1.711
X		X	X		X	81.22	3.136	89.54	2.013
	X	X	X		X	81.68	2.455	89.85	1.945
X	X		X	X		75.83	4.202	83.62	4.358
X	X		X		X	81.14	3.400	89.08	2.620
X		X		X	X	80.60	3.470	88.52	3.042
	X	X		X	X	81.49	2.473	89.42	2.118
		X	X	X	X	80.83	2.413	89.08	1.986
X			X	X	X	80.57	3.141	88.57	2.620
	X		X	X	X	81.57	2.468	89.71	1.879
X	X			X	X	81.11	2.550	89.06	2.070
X	X	X	X	X		75.00	3.049	83.11	2.490
X	X	X	X		X	80.76	2.680	88.90	1.927
X	X	X		X	X	79.72	3.984	87.97	2.775
X		X	X	X	X	80.60	3.278	88.38	2.851
	X	X	X	X	X	80.72	2.090	89.13	1.657
X	X		X	X	X	80.70	3.014	88.86	2.190

When combined with our previous results (those also presented in the original paper), a clear trend

is formed when all the signal combinations are plotted in a sorted-order with single signals, then two signals, then three, and so on.

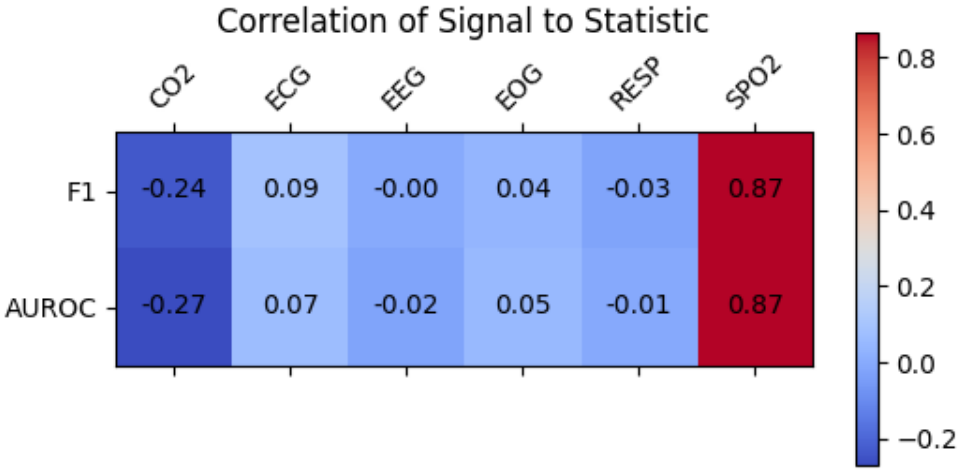


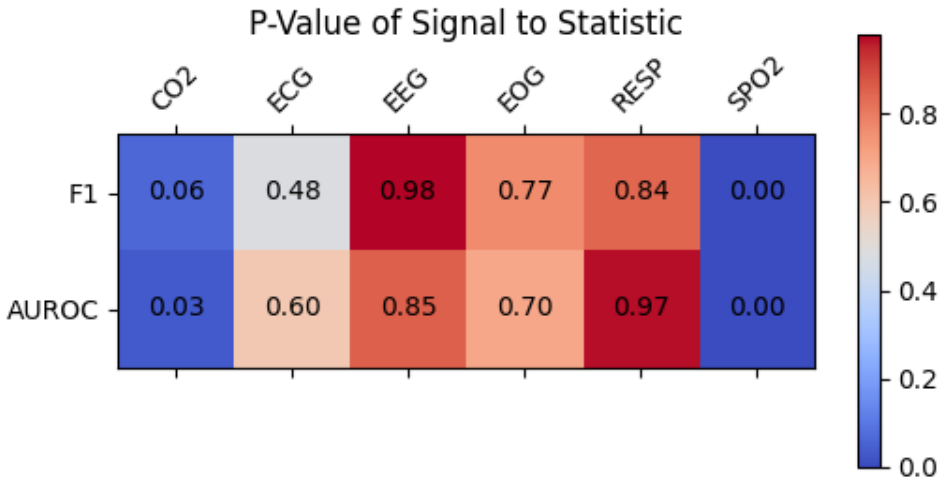
While some signals tend to be more clearly correlated with higher scores, the overall trend is that as the count of signals increases, generally speaking, the overall result is better.

10.5.2 Score Correlation of Individual Signals

Seeing the overall trends and that some signals were more "helpful" than others led us to ask whether there are signal combinations that perform better than the one selected in the paper (SPO2 and ECG).

To answer this question, we undertook to determine a Spearman rank correlation coefficient between the F1 and AUROC scores and the signals. This to the following results:





Based on these results and in terms of mean F1 score, SPO2 has the strongest correlation to a high score (correlation coefficient: 0.87) with ECG being a distant second (correlation coefficient: 0.09) and EOG being third (correlation coefficient: 0.04). The remaining signals demonstrate correlation with *lower* scores. AUROC scores follow the same pattern with slightly different correlation coefficients.

We can also see that the results of the SPO2 correlation are definitely non-random based on the p-score in that case of 0 for both F1 and AUROC. CO2 is definitely non-random for AUROC and for F1 it slightly exceeds the standard significantly-significant threshold of 0.05. CO2 is likely non-random due to the low p-value, despite that it does not meet the commonly accepted threshold. The other signals cannot disprove the null-hypothesis.

Overall, we can see that the original paper's selection of SPO2 and ECG captured the two largest contributors to high-scores. Our results showed that a combination of ECG, EEG, and SPO2 yielded the best results for F1 scores (81.88), but based on the p-value of the EEG signal for F1 scores (0.98), we refrain from assuming that this is would be true if trained on the entire NCH dataset due to the high likelihood that the EEG contribution is random.

11 Discussion

Since we were unable to acquire all the data used in the paper, and thus were not able to *exactly* reproduce the results outlined in the paper, we cannot assert with absolute certainty that this research is reproducible. However, the work we did with the data we were able to acquire strongly indicates the paper would be reproducible in its entirety, were we to have access to the entire dataset.

Further, the 63 total ablations (over 40 of which were beyond the scope of the paper) and additional statistical analyses we detailed in the previous section suggests the original research from the paper is sound.

As a result, we *believe* the research described in this paper is reproducible but cannot claim to know it is with complete confidence.

11.1 What was easy and what was hard

There was a substantial amount of open-source code written specifically to reproduce the findings in this paper. This code helped us make fast progress toward reproducing this research.

On the other hand, however, we did encounter a significant number of non-trivial issues in the process of reproducing this paper, but were ultimately able to surmount them. Some of these issues have been mentioned or implied previously, but are summarized as follows:

- Data necessary to reproduce this paper in its entirety are either impossible or very difficult, practically speaking, to acquire in a reasonable amount of time (less than a few weeks)
- Most of the code is undocumented
- Important parts of the codebase are non-functional, with any Python interpreter with which we're familiar
- Parts of the codebase included combination of parameters that are invalid
- Parts of their codebase do not match exactly the architecture discussed in the paper

11.2 Suggestions for improvement

We estimate that data acquisition problems are outside the authors' control, so we focus our suggestions for improvement herein on code quality. Below are the highest-impact tasks we would suggest the authors complete to improve their code and overall open-source repository for future contributors:

- Add clear documentation to each public Python function, including descriptions of parameters and return values
- Add Python type hints to at least the function parameters and return values of public functions
- Add complete information about the expected runtime environment, including required Python version(s) and dependency versions (including transitive dependencies)
  - We used Conda to manage this information and would suggest the authors use this system as well, if they don't already have a preferred system.
- Do an audit to ensure that model architecture and evaluations match those described in the paper
- Update their repository with changes as modifications are made during ongoing research
- Add at least minimal testing to ensure all code runs successfully. Ideas for tests include:
  - Construct the model, load it from pretrained weights, and do several inferences. Use this test to ensure the model runs with some minimal quality bar
  - Train the model for a small number of epochs to make sure loss is established and gradient descent/backpropagation works properly and loss begins decreasing
  - Preprocess and load one sleep study to ensure preprocessing/data loading code runs properly

11.3 Post-draft Plans for improvement

Between the draft due date and the final project due date, we planned to improve this notebook along several axes:

- Include more evaluations and visual aids to illustrate them (i.e. charts, graphs, etc…)
- Run at least one of the ablation studies from the paper and determine whether we reach the same conclusion as in the paper

We believe we met and surpassed these goals.

12 References

Fayyaz H, Strang A, Beheshti R. Bringing At-home Pediatric Sleep Apnea Testing Closer to Reality: A Multi-modal Transformer Approach. Proc Mach Learn Res. 2023;219:167-185.

Wei JL, Mayo MS, Smith HJ, Reese M, Weatherly RA. Improved behavior and sleep after adenotonsillectomy in children with sleep-disordered breathing. Arch Otolaryngol Head Neck Surg. 2007 Oct;133(10):974-9. doi: 10.1001/archotol.133.10.974. PMID: 17938319.

Wei JL, Bond J, Mayo MS, Smith HJ, Reese M, Weatherly RA. Improved behavior and sleep after adenotonsillectomy in children with sleep-disordered breathing: long-term follow-up. Arch Otolaryngol Head Neck Surg. 2009 Jul;135(7):642-6. doi: 10.1001/archoto.2009.74. PMID: 19620583.

Marcus, Carole L et al. "A randomized trial of adenotonsillectomy for childhood sleep apnea." The New England journal of medicine vol. 368,25 (2013): 2366-76. doi:10.1056/NEJMoa1215881

Lee H, Li B, DeForte S, et al. A large collection of real-world pediatric sleep studies. Sci Data. 2022;9(1):421. Published 2022 Jul 19. doi:10.1038/s41597-022-01545-6