Introduction

WAVi Medical manufactures lightweight and small-network EEG headsets, which have been used by many clinics in the United States to aid in the diagnosis of several neurological abnormalities, including epilepsy, ADHD, and concussion. EEG headsets collect voltage-signals through time at a set of electrode channels across the scalp. They are typically high-resolution in the temporal domain, with very little spatial resolution. The WAVi headset consists of 19 electrode channels spanning the scalp, and record voltage signals at a sampling rate of 250 Hz. The methods used to classify various disorders from EEG data vary wildly, and statistical algorithms which successfully define one phenotype may not work to significantly discriminate another. For example, power spectral density alone can often be used to tell when a patient is undergoing seizure activity. However, spectral density cannot be used to form an effective classifier of concussion, but pre- and post-injury coherence networks can be a powerful indicator. Often, we want to use the information of coherence networks to inform machine learning classifiers, or to interpret the features deep-learning methods may use to add discriminatory power. WAVi has collected EEG data from clinics specializing in the treatment of chronic skeleto-muscular pain, fibromyalgia, sports injury and concussion, opioid addiction rehabilitation, as well as a large log of over 2,500 reference population participants, with the goal of building informative classifiers aiding in the diagnosis and understanding of these conditions. Some conditions are better represented neurologically than others, and more insight is needed into several conditions before we can confidently apply our deep learning classifiers - namely chronic pain, fibromyalgia, and opioid addiction. These are of particular interest to the NIH, as the opioid epidemic in the United States continues to spiral out of control, and many clinicians

are looking for more effective ways to treat pain patients, and avoid prescribing opioids to those who may be experiencing pain as the result of withdrawal.

Results

For each time series in the dataset, we began by inspecting artifacted timepoints, and prevent the use of patients with poor quality data. Artifacted timepoints are exported along with the true time-series through the WAVi Desktop EEG collection software. The algorithm for artifacting timepoints is in-house, but can be boiled down to artifacting data that exceeds the typical standard deviation of the baseline signal, excessive voltage potentials, and abnormal coherence.

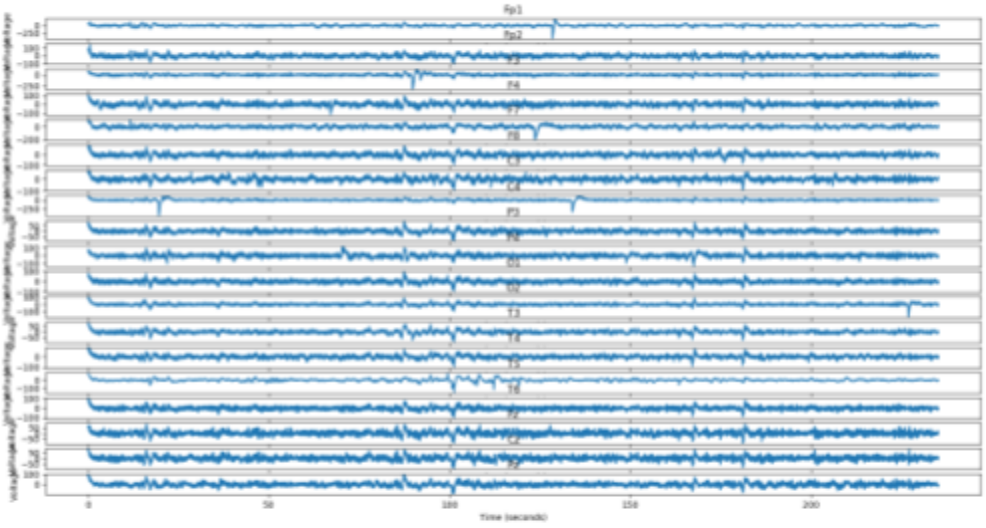

Figure 1A: Whole-trial time series representation of EEG data across 19 channels. Artifact-heavy timepoints are notable by their extremely-high potential. Most visible are those at ~165s and ~175s.

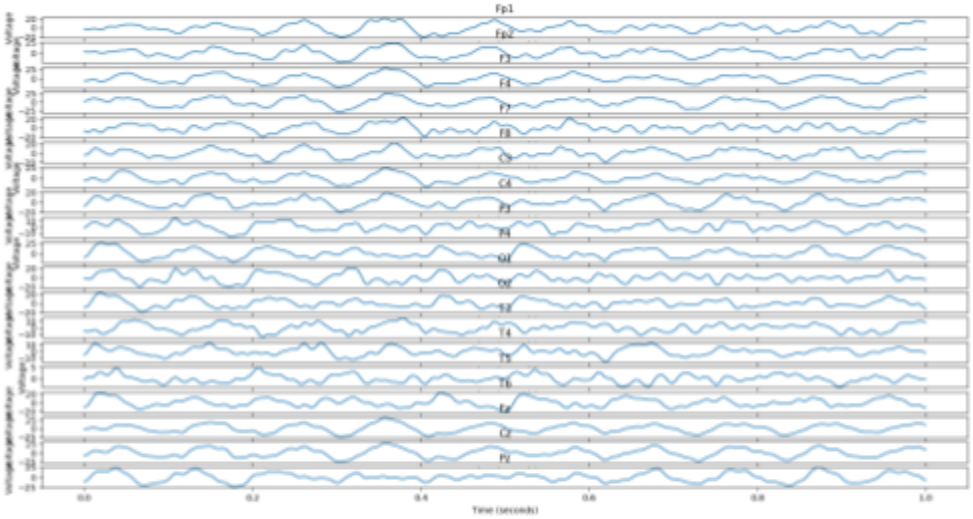

Figure 1B: 1-second contiguous time-series absent of WAVi-derived artifacted (noisy) timepoints spanning 19 channels across the scalp. This particular time window represents strong and easily recognizable power in the alpha (8-12 Hz) and beta (13-30 Hz) frequency bands, as well as recognizably-coherent signals across frontal and occipital electrodes.

Next, we fourier-transformed the data to inspect the resultant power-spectral density, and ensure that it meets standard criteria such as delta-offset, and strong alpha peaks, with relatively low noise in the 60 Hz range, which can indicate electrical interference.

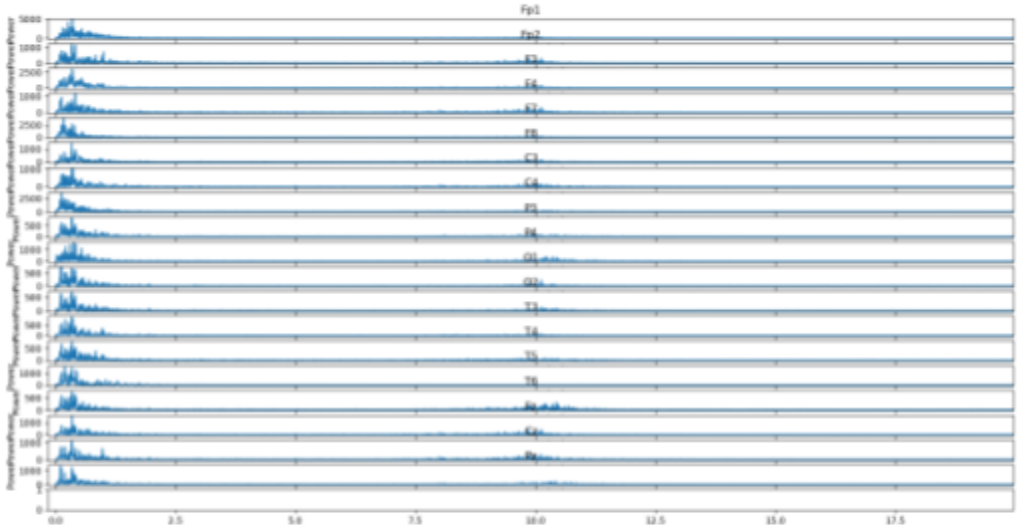

Figure 2: Whole-trial linear-scale spectra, indicating strong power in the 10 Hz range (alpha band)

The resultant coherence maps appear normal - though that's difficult to say without comparing them to the larger reference group. Initially, they tend to show weakening of coherence as frequency increases, and strongest coherence in the low-alpha to beta range.

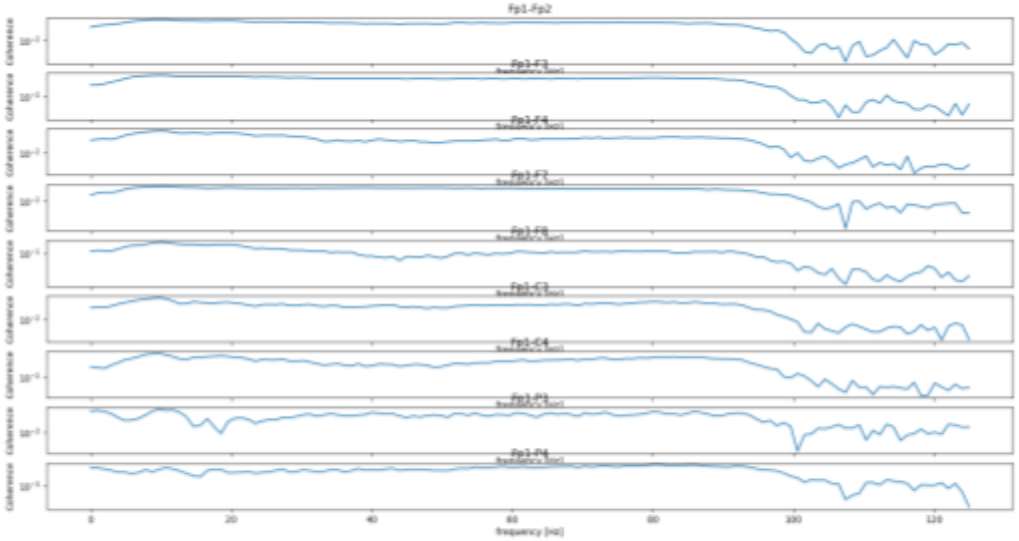

Figure 3: Example of first 6 of 171 total coherence values generated from whole-trial. The data in this figure was not filtered at any frequency. Again indicates strong alpha power, in addition to coherence components through the beta and gamma range.

The initial weighted network is not very informative. It shows varying degrees of coherence, typically strongest in the frontal regions (possibly due to muscle activity in the eyelids and eyebrows), generally weakening towards the parietal and occipital regions.

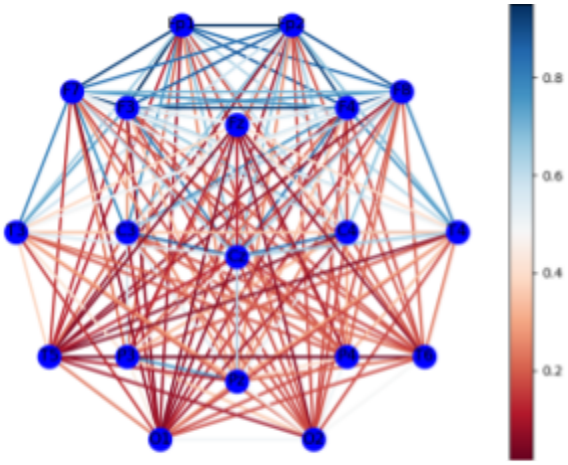

Figure 4A: Single-subject network representation of mean coherence signals generated in the alpha (8-12 Hz) band.

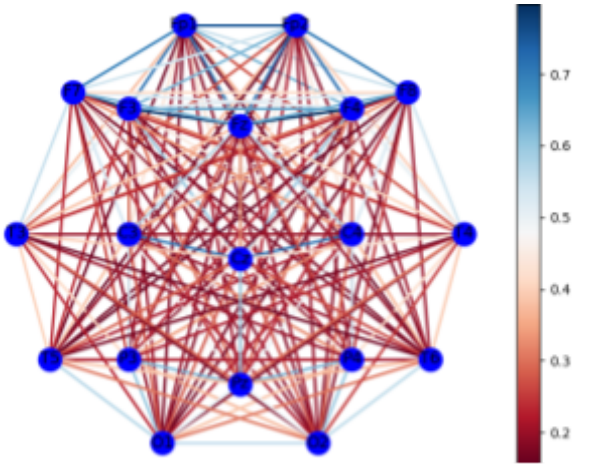

Figure 4B: Coherence averages in alpha band (8-12 Hz) for reference population group (N=1,669)

Once we have loaded in all the reference data, we use the distribution of coherence values for a given pair of nodes to calculate the z-score of another sample. Once we've done this for all the connections in a given dataset, we view them, drawing only those with a value exceeding that of the threshold z-score. When we provided the algorithm with withheld control-population data (i.e. healthy folks), no connections exceeded the z-score threshold of 1 in the alpha band. However, we saw extreme coherence values in both the chronic lower-back pain and opioid rehab datasets, which differed heavily from one another.

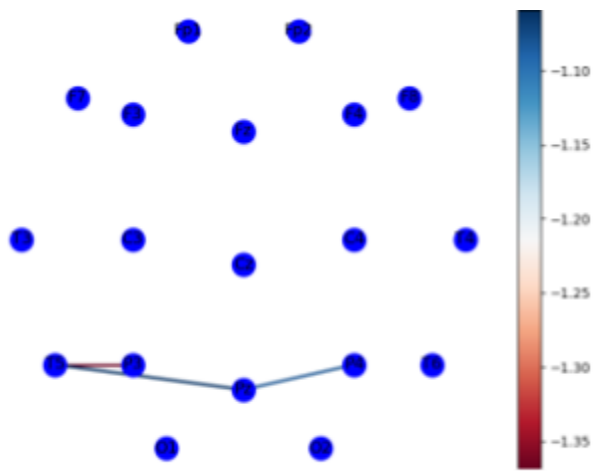

Figure 5A: Thresholded coherence map (z-score > 1) for chronic lower-back pain patients shows weaker-than-average coherence in parietal locations

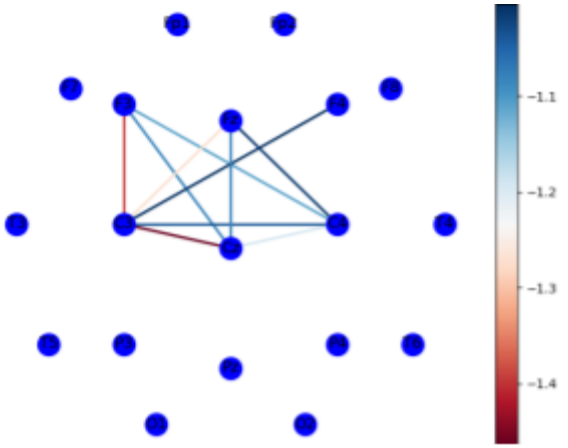

Figure 5B: Thresholded coherence map (z-score > 1) shows weaker-than-average connection between fronto-central electrodes in opioid addiction rehabilitation patients

Methods

To build coherence networks, we must first develop spectral components from the original time-series data. This is done through Fast Fourier Transforms (FFTs) (Bracewell, 1986), which are lossful representations of signal data in the fourier domain, rather than the temporal domain, represented by the following algorithm:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k n / N} \qquad k = 0, \ldots, N-1,$$

where x_0, ..., x_{N-1} are complex numbers, and $e^{-2\pi/N}$ is the primitive Nth root of 1. This formula is applied for each electrode of EEG, giving an array of 19 channels x 126 frequency bins. There is little evidence to show that there is useful discriminatory power in the higher frequencies of EEG, and these are often susceptible to interference from external electronic devices, but we allow the entire spectra to remain in our dataset until later in analysis. Once we have transformed the time-series data into the fourier-domain, we calculate the cross-power spectral density

(White, 1990). Given two signals $x(t)$and $y(t)$, each of which possess power spectral densities $S_{xx}(f)$and $S_{yy}(f)$, it is possible to define the cross power spectral density (CPSD) as:

$$P = \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} [x_T(t) + y_T(t)]^* [x_T(t) + y_T(t)]\, dt = \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} |x_T(t)|^2 + x_T^*(t) y_T(t) + y_T^*(t) x_T(t) + |y_T(t)|^2 dt$$

$$S_{xy}(f) = \lim_{T\to\infty} \frac{1}{T} [\hat{x}_T^*(f)\hat{y}_T(f)] \qquad S_{yx}(f) = \lim_{T\to\infty} \frac{1}{T} [\hat{y}_T^*(f)\hat{x}_T(f)]$$

This allows us to describe the coherence (magnitude-squared coherence) $C_{xy}(f)$ between the two signals $x(t)$ and $y(t)$ as:

$$C_{xy}(f) = \frac{|G_{xy}(f)|^2}{G_{xx}(f)G_{yy}(f)}$$

Which leaves us with T_{18} (18th triangular number) $= \frac{n(n+1)}{2} = 171$ representations of coherence, with a frequency range of 0 - 126 Hz. The frequency resolution of the resulting coherence signal is dependent on the sampling rate and length of the window supplied. In this analysis, I will describe the coherence of signals originating from EEG data collected for 4 minutes at 250 Hz. We ensure to retain the source locations of each paired coherence signal, for subsequent analysis.

Once coherence maps are generated for a given patient, they can be loaded into a dataset that summarizes the information of many patients at once. We do this by iteratively updating the mean, standard deviation, and count for each connection between node pairs, until the entire dataset has been loaded in. Once we have a reference coherence map, we can now refer to a map of patient-population coherences to this reference map, scoring the mean coherence of the patient map as a z-score on the normal distribution of the coherence for that particular connection. Using

varying z-score thresholds, we pop any non-extreme connections from the map, and show only the extreme values exceeding the threshold.

Discussion

The results of the network coherence maps have shown us useful insights into the "connectome" of EEG for the two patient populations studied: chronic pain and opioid rehabilitation. These insights will hopefully prove useful in building discriminators for these two conditions - a task considered very important to public health at the moment. Over 1 billion people worldwide suffer with chronic pain, and it's important that we begin treating them effectively without allowing our treatments' adverse effects to spiral out of control. I am curious to know how the exclusion of the relevant electrodes from deep-learning analysis will affect the current classification score. We would expect that the score will decrease, of course, but whether they will decrease more heavily for the most relevant coherence connections is an important question that can inform us about the power of convolutional neural networks and other deep learning methods.

Bibliography

Bracewell, R. N., & Bracewell, R. N. (1986). The Fourier transform and its applications (Vol. 31999). McGraw-Hill New York.

L. B. White and B. Boashash, "Cross spectral analysis of nonstationary processes," in IEEE Transactions on Information Theory, vol. 36, no. 4, pp. 830-835, July 1990, doi: 10.1109/18.53742.

Appendix: Code

https://github.com/canlab/WAViMedEEG/

Networks.py

```
import networkx as nx
import numpy as np
import os
from src import config
from src import Standard
from matplotlib import pyplot as plt

class Coherence:
    def __init__(self, labels=config.channel_names):

        self.labels = labels
        # add each channel as fully-connected node
        self.G = nx.complete_graph(self.labels)
        self.edgelist = [edge for edge in self.G.edges()]
        for edge in self.edgelist:
            self.G.edges[edge]['n'] = 0
            self.G.edges[edge]['mean'] = 0
            self.G.edges[edge]['sum'] = 0
            self.G.edges[edge]['sum2'] = 0
            self.G.edges[edge]['std'] = 0

        # print("Edge list:", self.edgelist)

        self.coherences = []

        self.f = None

    def write(self, path):
        nx.write_gpickle(self.G, path)
```

```
def load_network(self, path):
    self.G = nx.read_gpickle(path)

def load_data(self, path):
    for fname in os.listdir(path):
        arr = np.genfromtxt(path+"/"+fname, delimiter=",")
        coh = Standard.CoherenceMap()
        coh.subject = fname.split('_')[0]
        coh.task = fname.split('_')[1]
        coh.map = arr[1:]
        coh.f = arr[0]
        if self.f is None:
            self.f = coh.f
        self.coherences.append(coh)

def score(self, band="alpha"):
    self.band = band
    min, max =\
    config.frequency_bands[self.band][0],\
    config.frequency_bands[self.band][1]

    Cxy_range = np.where((self.f >= min) & (self.f <= max))[0]
    Cxy_min, Cxy_max = Cxy_range[0], Cxy_range[-1]

    for coh_map in self.coherences:
        for Cxy, edge in zip(coh_map.map, self.edgelist):
            coh_map.coherence_value = np.mean(Cxy[Cxy_min:Cxy_max])
            self.G.edges[edge]['n'] += 1
            self.G.edges[edge]['mean'] =\
                (self.G.edges[edge]['mean']*(self.G.edges[edge]['n'] - 1)\
                + coh_map.coherence_value) / self.G.edges[edge]['n']
            self.G.edges[edge]['sum'] += coh_map.coherence_value
            self.G.edges[edge]['sum2'] +=\
                (coh_map.coherence_value - self.G.edges[edge]['mean'])**2
            self.G.edges[edge]['std'] = np.sqrt(
                self.G.edges[edge]['sum2'] / (self.G.edges[edge]['n']))
    self.edgelist = [edge for edge in self.G.edges()]

def draw(self, weighting=True, threshold=False):
```

```
    for node, pos in zip(
        [node for node in self.G.nodes()], config.networkx_positions):
        self.G.nodes[node]['pos'] = pos

    if (weighting is True) and (threshold is False):
        edges, weights = zip(*nx.get_edge_attributes(self.G,'mean').items())
    if (weighting is True) and (threshold is True):
        edges, weights = zip(*nx.get_edge_attributes(self.G, 'z-score').items())
    else:
        weights = [1 for edge in self.G.edges()]
        edges = [edge for edge in self.G.edges()]

    pos = nx.get_node_attributes(self.G, 'pos')

    cmap=plt.cm.RdBu
    vmin = min(weights)
    vmax = max(weights)

    nx.draw(
        self.G,
        pos,
        with_labels=True,
        node_color='b',
        edgelist=edges,
        edge_color=weights if weighting is True else None,
        width=2.0,
        edge_cmap=cmap,
        vmin=vmin,
        vmax=max)

    if weighting is True:
        sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(
            vmin = vmin, vmax=vmax))
        sm._A = []
        plt.colorbar(sm)

    plt.show()

def threshold(self, reference=None, z_score=2):
```

```
        if reference is None:
            reference = self.G
        means = [val[1] for val in nx.get_edge_attributes(reference,'mean').items()]
        stds = [val[1] for val in nx.get_edge_attributes(reference,'std').items()]

        for i, edge in enumerate([edge for edge in self.G.edges()]):
            z = (self.G.edges[edge]['mean'] - means[i]) / stds[i]
            print(z)
            if np.abs(z) < z_score:
                self.G.remove_edge(edge[0], edge[1])
                print("Removing:", edge)
            else:
                self.G.edges[edge]['z-score'] = z
```

Run_coherence_analysis.py

```
import sys
sys.path.append('..')
from src import Prep
from src import Standard
from src import Networks
from src import config
import os
from tqdm import tqdm
import argparse


def main():

    parser = argparse.ArgumentParser(
        description='Options for Standard.BandFilter method')

    parser.add_argument('--studies_folder',
                        dest='studies_folder',
                        type=str,
                        default=config.my_studies,
                        help="(Default: " + config.my_studies + ") Path to "
                        + "parent folder containing study folders")

    parser.add_argument('--reference_study',
```

```
                dest='reference_study',
                type=str,
                default=None,
                nargs='+',
                help='(Default: None) Study folder containing '
                + 'reference dataset. ')

parser.add_argument('--study_name',
                dest='study_name',
                type=str,
                default=None,
                nargs='+',
                help='(Default: None) Study folder containing '
                + 'dataset. '
                + 'If None, performed on all studies available.')

parser.add_argument('--task',
                dest='task',
                type=str,
                default="P300",
                help="(Default: P300) Task to use from config.py: "
                + str([val for val in config.tasks]))

parser.add_argument('--type',
                dest='type',
                type=str,
                default="bandpass",
                help="(Default: bandpass) Which band filter method "
                + "should be applied: "
                + "lowpass, highpass, bandstop, bandpass")

parser.add_argument('--band',
                dest='band',
                type=str,
                default="alpha",
                help="(Default: alpha) "
                + "Frequency band used for band ranges: "
                + str([val for val in config.frequency_bands]))

# save the variables in 'args'
```

```
args = parser.parse_args()

studies_folder = args.studies_folder
reference_study = args.reference_study
study_name = args.study_name
task = args.task
type = args.type
band = args.band

# if reference_study is None:
#     ref_studies = os.listdir(studies_folder)
# else:
ref_studies = [study+"/coherences/"+task for study in reference_study]

# if study_name is None:
#     my_studies = os.listdir(studies_folder)
# else:
my_studies = [study+"/coherences/"+task for study in study_name]

# ERROR HANDLING
if not os.path.isdir(studies_folder):
    print(
        "Invalid entry for studies_folder, "
        + "path does not exist as directory.")
    raise FileNotFoundError
    sys.exit(3)

if ref_studies is not None:
    for study in reference_study:
        if not os.path.isdir(os.path.join(studies_folder, study)):
            print(
                "Invalid entry for reference_study, "
                + "path does not exist as directory.")
            raise FileNotFoundError
            sys.exit(3)

if my_studies is not None:
    for study in study_name:
        if not os.path.isdir(os.path.join(studies_folder, study)):
            print(
```

```
                "Invalid entry for study_name, "
                + "path does not exist as directory.")
            raise FileNotFoundError
            sys.exit(3)

    if task not in config.tasks:
        print(
            "Invalid entry for task, "
            + "not accepted as regular task name in config.")
        raise ValueError
        sys.exit(3)

    if type not in ["lowpass", "highpass", "bandpass", "bandstop"]:
        print(
            "Invalid entry for type, "
            + "must be one of: lowpass, highpass, bandpass, bandstop")
        raise ValueError
        sys.exit(3)

    if band not in config.frequency_bands:
        print(
            "Invalid entry for band, "
            + "must be one of: " + [val for val in config.frequency_bands])
        raise ValueError
        sys.exit(3)

    ref_coh = Networks.Coherence()
    for study in ref_studies:
        ref_coh.load_data(studies_folder+"/"+study)
    ref_coh.score(band=band)
    # ref_coh.draw(weighting=True)

    study_coh = Networks.Coherence()
    for study in my_studies:
        study_coh.load_data(studies_folder+"/"+study)
    study_coh.score(band=band)
    study_coh.threshold(reference=ref_coh.G, z_score=1)
    study_coh.draw(weighting=True, threshold=True)

if __name__ == '__main__':
```

main()