Designing Convolutional Neural Networks and Autoencoder Architectures for Sleep Signal Analysis

by

Michael Sokolovsky

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

April 2018

APPROVED:

______________________________
Professor Carolina Ruiz, Thesis Advisor

______________________________
Professor Sergio A. Alvarez (Boston College), Thesis Co-Advisor

______________________________
Professor Xiangnan Kong, Thesis Reader

______________________________
Professor Craig E. Wills, Head of Department
Abstract

In this thesis we explore how Deep Convolutional Neural Networks (CNNs) can be used for increasing the efficiency of current sleep stage classification methods and for uncovering new signal features in sleep study data. Polysomnograms (PSGs) are multi-parameter tests composed of EEG and other time-varying sleep signal data gathered from patients during sleep studies. Using the public Physionet Database of PSG data, we apply recent advancements in deep learning to design, build and train a supervised CNN model to classify sleep signals into sleep stages. Our model yields across-validated classification accuracy of 81%, which is on par with the performance of an ensemble of human experts and outperforms prior work using shallow CNNs. Our work demonstrates that it is possible to create accurate hypnograms from PSG data automatically using CNNs. Furthermore, we explore the application of deep CNNs for feature discovery in PSG data. We train Deep Convolutional Autoencoders on PSG data and project the data into new feature spaces using the trained encoders. We discover that clustering data in the encoded projection spaces does not yield useful information from PSG data. We propose and implement a new iterative-deepening algorithm called bow tie training for training autoencoders. We show that bow tie training performs as well as current autoencoder training methods at minimizing data reconstruction loss, and we demonstrate that our bow tie method achieves faster results at training ensembles of autoencoders.
Acknowledgements

Thank you, Professors Carolina Ruiz and Sergio Alvarez, for your guidance throughout the research and writing process of my thesis. I entered WPI with little research experience and only the beginnings of an understanding of machine learning. It was through your direction that I now have a grasp on modern approaches to machine learning that two years ago seemed like black magic. Thank you for your part in getting me to where I am.

Thank you, Andres Francisco Guerrero, Sarun Paisarnsrisomsuk, Kabir Ahmedul, and Sam Ogden for your camaraderie. Thank you for talking with me honestly about not only school or computer science but also subjects beyond our work at WPI. Thank you for your willingness to engage with me as fellow humans trying to live ethically in a complex world rather than just as graduate students.

Thank you, Ben Schreiber for continuing to play music with me, God only knows why. You understand comedic and musical parts of my brain that no one else does. Without our band Uncle Farmer, slogging through the particularly grueling bits of this exercise would have been unbearable. Thank you, Susan Petrick, for taking us on adventures and being the coolest road mom we could ask for. Thank you, Sam Biagetti and Joe Maurer. Thank you, Mara Feingold-Link.

Thank you, family, Vitaly, Elina, and Eugene, for your support and for bringing perspective to my life. I do not doubt that my knack for critical thinking comes from you.
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>5</td>
</tr>
<tr>
<td>List of Tables</td>
<td>6</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>2 Background</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Polysomnography</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Sleep Scoring</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Convolutional Neural Networks</td>
<td>10</td>
</tr>
<tr>
<td>2.4 Autoencoders</td>
<td>12</td>
</tr>
<tr>
<td>2.5 Autoencoder Training</td>
<td>12</td>
</tr>
<tr>
<td>3 Sleep Stage Classification</td>
<td>15</td>
</tr>
<tr>
<td>3.1 Methodology</td>
<td>16</td>
</tr>
<tr>
<td>3.1.1 Data</td>
<td>16</td>
</tr>
<tr>
<td>3.1.2 Model Architecture</td>
<td>16</td>
</tr>
<tr>
<td>3.1.3 Training</td>
<td>18</td>
</tr>
<tr>
<td>3.1.4 Evaluation</td>
<td>18</td>
</tr>
<tr>
<td>3.2 Results</td>
<td>19</td>
</tr>
<tr>
<td>3.3 Discussion</td>
<td>21</td>
</tr>
<tr>
<td>4 Sleep Signal Feature Extraction</td>
<td>22</td>
</tr>
<tr>
<td>4.1 Methodology</td>
<td>23</td>
</tr>
<tr>
<td>4.1.1 Autoencoder Architectures</td>
<td>23</td>
</tr>
<tr>
<td>4.1.2 Autoencoder Training</td>
<td>25</td>
</tr>
<tr>
<td>4.1.3 Clustering Experiments</td>
<td>28</td>
</tr>
<tr>
<td>4.1.4 Supervised Learning Experiments</td>
<td>29</td>
</tr>
<tr>
<td>4.2 Results</td>
<td>29</td>
</tr>
<tr>
<td>4.2.1 Autoencoder Training Experiments</td>
<td>29</td>
</tr>
<tr>
<td>4.2.2 Encoder Clustering Experiments</td>
<td>35</td>
</tr>
<tr>
<td>4.2.3 Supervised Learning Experiments</td>
<td>39</td>
</tr>
<tr>
<td>4.3 Discussion</td>
<td>41</td>
</tr>
<tr>
<td>5 Conclusions and Future Work</td>
<td>43</td>
</tr>
<tr>
<td>Bibliography</td>
<td>44</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Polysomnogram and Hypnogram Illustration .................... 10
2.2 Convolutional Neural Network Illustration ...................... 10
2.3 Illustration of Fully-Connected Autoencoder ................... 13
2.4 Autoencoder Layer-Wise Training ................................. 14

3.1 Network Architecture ............................................. 17

4.1 Example of Autoencoder .......................................... 25
4.2 Layer-Wise Training ................................................ 26
4.3 Bow Tie Training .................................................... 27
4.4 Autoencoder Reconstruction Examples ............................ 32
4.5 Bow Tie Validation Loss ............................................ 32
4.6 Validation Loss Comparisons ..................................... 34
4.7 MDS Projections: Raw Signal vs. Spectrogram .................. 37
4.8 MDS Projections: Spectrogram Input vs. Output ............... 38
List of Tables

3.1 Cumulative Confusion Matrix .................................. 19
3.2 Metric Bounds .................................................... 19
3.3 Inter-Scorer Agreement ........................................... 20

4.1 Layer-Wise Training Times ..................................... 31
4.2 Bow Tie Training Times ......................................... 31
4.3 Encoder Decision Tree Performance ........................... 40
Chapter 1

Introduction

In this thesis, we explore how Deep Convolutional Neural Networks (CNNs) can be used for increasing the efficiency of current sleep stage classification methods and for uncovering new signal features in sleep study data. Polysomnograms (PSGs) are multi-parameter tests composed of EEG and other time-varying sleep signal data gathered from patients during sleep studies. PSG data is crucial for diagnosing sleep-related illnesses because of their use in sleep stage classification; technicians map PSG data to sequences of standard sleep stages known as hypnograms, which are then used by clinicians to make diagnostic decisions. Mapping PSG data to sleep stages is a time-consuming process that could be aided by modern machine learning approaches. Furthermore, sleep stages are human constructs; though sleep staging help doctors in making diagnoses, the sleep stage paradigm may miss out on capturing other diagnostic information present in PSG data. The goal of our research is to address these problems by exploring how deep CNNs can be used for both automatic sleep stage classification for uncovering new features in PSG data.

CNNs are often used for image classification and two-dimensional pattern detection tasks. However, less research has been done in one-dimensional signal classification with deep networks. Prior research exists in sound recognition [1, 2, 3] and EEG signal analysis [4]. Deep neural network research in sleep stage classification from PSG signals is rarer. We are aware of prior works using autoencoders [5], deep belief networks [6], and CNNs [7], respectively.

Using the public Physionet Database of PSG data, we apply recent advancements in deep learning to design and train a supervised CNN model to classify sleep signals from 20 patients’ PSG data into sleep stages. Our model yields a cross-validated classification accuracy of 81%, which is on par with the performance of an ensemble of human experts and outperforms prior work using shallow CNNs [7]. Our work demonstrates that it is possible to create accurate hypnograms from PSG data automatically using CNNs.

Furthermore, we explore the application of deep CNNs for feature discovery in PSG data. We train Deep Convolutional Autoencoders on PSG data and project the data into new feature spaces using the trained encoders. We discover that clustering data
in the encoded projection spaces does not yield useful information from PSG data. We propose and implement a new iterative-deepening algorithm called bow tie training for training autoencoders. We show that bow tie training performs as well as current autoencoder training methods at minimizing data reconstruction loss, and we demonstrate that our bow tie method achieves faster results at training ensembles of autoencoders.

Our research contributions are outlined as follows. We create a CNN model for classifying sleep signal data that performs as well as human experts; applications of our work could be used for improving the efficiency of creating hypnograms from PSG data. We additionally propose a new autoencoder training method. We demonstrate that this training method performs as well as current methods at minimizing reconstruction loss and improves on training times when training ensembles of models.
Chapter 2

Background

Polysomnography

Sleep is vital to human health. Sleep deficiency correlates with health problems including Parkinson’s disease [8] and heart disease [9, 10], and polysomnographic sleep studies provide an important tool in diagnosing sleep disorders associated with sleep deficiency. Polysomnography is a multi-parameter test used for diagnosing sleep-related illnesses; electrical signals from sensors placed on the body during sleep, collectively called polysomnograms or PSGs, are recorded and used for analyzing patients’ sleep. PSGs are composed of data from electroencephalograms (EEG, corresponding to cortical electrical activity), electrooculograms (EOG, corresponding to movement of the eyes), electromyograms (EMG, corresponding to movement of the chin), electrocardiograms (ECG, corresponding to electrical activity of the heart), as well as other signals corresponding to respiration air flow and blood oxygen levels.

Sleep Scoring

The American Academy of Sleep Medicine (AASM) maintains standards for conducting sleep studies and for categorizing PSG signal data into four sleep stage categories: Sleep Stage 1 (S1), Sleep Stage 2 (S2), Sleep Stage 3 (S3) [11], and Rapid-Eye Movement or REM (R). Stages 1–3 describe varying depths of sleep. Stages 1 and 2 are often termed light sleep (with Stage 1 being the lighter of the two), while Stage 3 is termed deep sleep. REM sleep owes its name to rapid movements of the closed eyes during sleep, and is associated with dreaming [12], though dreams are now known to occur in other stages of sleep as well.

During staging, PSG signals are divided into 30-second intervals called sleep epochs, each of which is scored by a technician into either one of the four stages, a wake stage, or a movement stage. The resulting sequences of sleep stages are known as hypnograms. The process of mapping a signal from a PSG to a sleep stage is illustrated in fig. 2.1.
Scoring decisions are made by quantitative and visual analysis of electrical signals, relying on spectral characteristics such as low-frequency (delta band) waves in stage S3, as well as time-domain features such as K-complexes in stage S2. Sleep technicians must be trained, and the task of scoring a sleep study requires considerable effort and time.

Convolutional Neural Networks

Convolutional Neural Networks or CNNs are variants of neural network statistical learning models which have been successfully applied to image recognition tasks, achieving current state-of-art results in image classification [13, 14]. Like previous neural networks architectures, CNNs are hand-built models whose inputs are unlabelled data in the form of vectors or tensors and whose outputs provide information about the unknown labels associated with the data, such as the data’s probability of belonging to a certain class.

CNNs are composed of stacks of three main types of processing layers: convolutional layers, pooling layers, and fully connected layers. Each successive layer applies a function to its input data tensors and passes a transformed version of the data tensors.
as input to the next layer. An example of a CNN model is illustrated in fig. 2.2, and described below are each of the types of layers present in CNN models:

**Convolutional Layers:** Convolutional layers apply linear transformations followed by non-linear activation functions to obtain each individual component of the data vector to be passed to the next layer. The components of a layer’s input vector that contribute to a given output component are restricted to a small portion of the input layer, making the linear transformation a ”local” operation. Furthermore, the coefficients of these linear transformations are identical for all output components, making processing invariant to translations in the input.

**Pooling Layers:** In contrast with convolutional layers, pooling layers simply down-sample the data vectors from the preceding layer by applying an aggregating function, yielding a lower-resolution rendering of the input vectors. Pooling is intended to control over-fitting by reducing the total number of parameters in the network, and to increase robustness to minor variations in the input data.

**Fully Connected Layers:** The final output layers of a CNN are most often fully-connected layers as in traditional neural networks. A fully connected layer applies a linear transformation to its input vectors, followed by a nonlinear activation function. Unlike a convolutional layer, the linear transformation associated with a fully connected layer is not subject to the locality and spatial invariance constraints. Thus the output of a fully connected layer is the result of applying the nonlinear activation function to a general linear transformation of the full input vector.

Neural networks’ representational power stems from their ability to represent highly nonlinear functions through the composition of nested nonlinear activation functions and linear transformations. While traditional neural network layers apply functions to their entire input vectors, the layers in CNNs apply functions to smaller local patches of data. This can be interpreted as taking advantage of certain independence assumptions on the features in the input data–namely, that features spatially close to one another have more relevance to each other than features far from one other. The application of functions to local patches of data occurs in the convolutional layers, described above. The sharing of weights across a local patch function can also be viewed equivalently as a sliding ”filter,” or kernel, moving across the input; this can be thought of as corresponding to a sensor looking for a specific pattern by sliding across the input tensor. In practice, each convolutional layer uses multiple filters with different weights, each applied to the input data. For example, if there are $n$ filters in a convolution layer, the output should have a dimension $n$ as well, representing the outputs of each filter sliding over the input data. Although use of filters applied to local patches requires fewer parameters and therefore decreases the representational power of CNNs compared to fully-connected neural networks, they also reduce the computation time needed for training models in domains with high-dimensional data.

Due to the presence of these filters, CNNs work well in domains where local patterns are meaningful for classification. An example in the domain of sleep scoring is the importance of translation-invariant signal features, such as K-complexes and sleep spindles [11]. Prior work in visualizing CNN layer outputs suggest that CNNs extract
local patterns within a data sample in early layers and aggregate them into larger patterns descriptive of the entire sample in deeper layers of the model [15]. This parallels the methods that sleep technicians use in evaluating static sections of PSGs for sleep classification.

Autoencoders

As discussed in chapter 3, neural networks, when trained in a supervised manner, have an input layer that takes in the features of data and an output layer that encodes a prediction of the class. The final layer can take the form a vector whose values represent the probability of certain classes. Parameters of the network are then trained using some variant of stochastic gradient descent on batches of data. Autoencoders, first described in [16] are a variant of neural networks that train to reconstruct the original input features to its network rather than to make class predictions; for dataset $D$ whose data have dimension $n$, an autoencoder will have an input and output layer of dimension $n$ and a target output comprised of the same data in $D$.

With this framework, a neural network can be trained in an unsupervised fashion using only the dataset. The usefulness of a such a network is less in its ability to reconstruct data and more in its ability to reduce the dimensionality of the data and in this process, project the data into a new feature space. If the layers between the input and output layers have the same or lower dimensionality than the inner layers, the optimal weight configuration of the network would yield a simple identity mapping. However, training an autoencoder that has internal layers of lower dimensionality than the input and output layers forces the network to update its weight in such a way that it learns to compress data. Thus, internal layers train to function as maps from the original data to a compressed lower-dimensional projection. Prior research confirms this behavior; autoencoders with linear activations learn data representations that are similar to PCA representations [17]. Autoencoders with nonlinear activations, and variants such as convolutional autoencoders have the ability to construct more complex feature spaces.

Autoencoder Training

Autoencoders are composed of two sections: an encoder that maps input data to a lower-dimensional representation and a decoder that maps the output of the encoder to the same space as the input data. Classic autoencoders have sequentially decreasing and increasing dimensionality of layers in the encoder and decoder, respectively. As illustrated in fig. 2.3, autoencoders usually are built to have symmetrical encoder and decoder layers.

Autoencoders are typically trained in a greedy, layer-wise fashion using multiple shallow autoencoders. First, a shallow autoencoder trains on the original input data. New successive shallow autoencoders train on the outputs of previously-trained shallow encoders until a desired depth is reached. Finally, a full-size autoencoder is assembled from the small autoencoders and trained on the original data. The development of this
method arises from the observation that stochastic gradient descent methods starting from random initializations often fall into poor local minima, especially in the presence of bottlenecks in networks [18]. Training smaller subsets of the networks tends to mitigate the problem and lead to lower reconstruction error [18][19][20]. Layer-wise training additionally leads to faster training times; the smaller autoencoders have fewer weights so values for these weights converge to local minima faster, and the pre-training of these models initialize the larger, assembled autoencoder’s weights in the parameter space close to local minima, leading to a fast final training run.

Section 2.5 outlines this layer-wise training method; an autoencoder composed of an input, output, and two smaller inner layers train on the data set. Successive autoencoders, where the input and output layers are the previous autoencoders’ inner layers, train on the data passed through the previously-trained ensemble of encoders. Once a desire depth of an encoder is reached, the ensemble of trained autoencoders are assembled into a full-size autoencoder and trained again on the full dataset.
Figure 2.4: Illustration of layer-wise training of an autoencoder with a depth of 4
Chapter 3

Sleep Stage Classification

Over the last decade, deep convolutional neural networks, or CNNs, have demonstrated state-of-the-art classification results in image categorization [21, 14]. Their development stems from the insight that the features comprising images, the pixel values, are highly dependent on the values of their neighbors, but have little dependence or relevance to pixels farther away. For example, for an object in a picture to be of an elephant, one would expect pixels inside the object to be grey. What color the ground, pavement, or grass the object is on has no bearing on whether the object is an elephant. CNNs take advantage of this property by applying linear functions to local patches in images, which allows for fewer parameters, faster training times, and thus more deep models than their fully-connected neural network counterparts.

Though the literature on CNNs’ applications for image processing is rich, CNNs applications for signal processing have received less attention. This may be due to the fact that signal processing challenges often lend themselves to model frameworks different from the assumptions implicit to CNNs; in many signal processing problems the goal is not to classify an entire signal of discrete values but rather to predict attributes of the signal at certain points in time. Yet there remains a realm of problems where thinking of signals as one-dimensional “images”, with local patches of highly-correlated information, is a natural approach to viewing the data.

Sleep signal data and its classification, the subject of the first phase of our project work, is one example of a type of data and problem that lends itself to this framework and thus to being solved with CNNs. Sleep stage classification is an important task in clinical medicine in which human experts map EEG and other time-varying sleep signal data to a sequence of standard sleep stages known as hypnograms; this visual signal processing technique is essential to diagnosing sleep-related illnesses [22]. The technique relies on highly-trained human technicians and is prone to error. Automated sleep scoring can contribute to more efficient and reliable diagnosis of sleep-related disorders.

In the first phase of our project, we design and evaluate a deep CNN architecture that performs automatic sleep stage classification of human sleep signals using EEG and EOG signal channels as inputs. Our methodology is presented in section 3.1 alongside
our results in section 3.2. We present a concluding remarks about our findings in section 3.3.

Methodology

Data

We use PSGs from the publicly-available Physionet database [23] for training and evaluating neural network models. Specifically, we use the Study 1 data from the Sleep-EDF Database [Expanded] [24]. This database is composed of 20 patients’ PSG data from two full days of recording, totaling 39 recordings (data from one patient was only available for one day). PSG data for each patient consisted of two EEG signals, (EEG Fpz-Cz, EEG Pz-Oz) and one EOG signal (EOG horizontal) sampled at 100Hz. Accompanying hypnograms for the full day PSG recordings are included, with each day of recording scored by one of six human expert scorers. To avoid including epochs from non-sleep periods, raw data between the first observed epoch of sleep and the last observed epoch of sleep are extracted from each study and preprocessed as described below.

Epoch classification depends on neighboring epoch information [11]. To mimic exploration of these features in our network, we preprocess multi-hour PSG signals into 150-second, or 5-epoch, samples as in [7]. Models then train to categorize the middle epoch of each sample. The movement epochs during sleep are removed from the dataset prior to training due to their rarity, leaving five options of sleep encodings for each sleep epoch. Given that three signal channels are used, two EEG channels and one EOG channel, input data to the network takes the form of two-dimensional data of shape (15000,3) composed of three 150-second signals sampled at 100Hz. The first convolutional layer in the model interprets the three signals as channels of a 15,000-length vector. The class labels are coded as one-of-five categories (S1, S2, S3, EEG, Awake) in a one-hot encoded vector.

Model Architecture

The model architectures we explore for our project are inspired by recent state-of-the-art networks which achieve high performance by increasing depth. We test architectures modeled on the VGG network [14] which featured small convolutional filters, multiple stages with stacked convolutional layers separated by pooling layers, and increasing numbers of filters with depth, alongside variants of residual networks [25] featuring skip connections. The model we report is the current best-performing model, based on the architecture suggestions outlined in [26] and work conducted in [27]. In [27], the author modifies our architectures and finds that applying less-aggressive pooling layers in final layers leads to performance improvements. Our final model reflects these findings. This model structure is shown in fig. 3.1 and described below.

Following [26], the model is structured in levels containing stacked convolutional layers separated by pooling layers. The model contains seven levels with 6 convolutional layers in the first level, 3 convolutional layers in each of the three levels that follow,
a single convolutional layer in the fifth and sixth levels, followed by a final sixth level consisting of two fully-connected layers. All early convolutional layers have kernel sizes of length 100 as in [7] and stride size of 1. Because the sleep epoch to be classified is in the middle of each 150-second input vector, models use no padding during training, and experiments with padding did not yield better results. As in [14], the number of filters in each convolutional layer stays the same or increases after down-sampling so that each layer within every stage requires roughly the same computational time. Activations in all layers are ReLu units, and the output of the model is a vector of five numbers representing the probability of each class, calculated via a final softmax layer.

Figure 3.1: Network Architecture, as described in section 3.1.2
Training

We evaluate models using two tiers of cross-validation; models first train using four-fold cross validation, and the best-performing model from that set is then retrained using ten-fold cross validation. For the first tier of training, the twenty patients’ data are compiled into four folds, each containing training, validation, and testing sets: test sets include five patients, validation sets include two patients, and training sets include thirteen patients. Folds are randomly compiled so that every patient’s data appear exactly once in a testing set for one of the four folds. Similar to the first training procedure, in the second tier of training, ten folds are randomly compiled so that every patient’s data appears exactly once in a testing set for one of the folds. Test sets contain two patients, validation sets contain three patients, and training sets contain fifteen patients.

We initialize model weights as in [25], by sampling from a Gaussian distribution with zero mean and standard deviation $\sqrt{2/n_l}$ in layer $l$, where $n_l$ is the product of the number of input channels and the number of weights per filter in layer $l$. The models train with stochastic gradient descent on batches of size 256 to minimize the cross entropy loss function applied to the softmaxed output. To account for class imbalances, we weight gradient updates from mini batch samples by the inverse of class frequency in the training set to achieve balanced training across each class. The only regularization present during training is the addition of batch normalization layers preceding nonlinear activations as in [28]. The initial learning rate of 0.01 is set to progressively decrease after validation accuracy stops increasing. Models train for between 30 and 100 epochs. For ten-fold experiments, training sets consist of approximately 27,000 samples. We build all models in TensorFlow [29] and Keras [30] running on the NVIDIA CUDA platform, using NVIDIA K20 and K80 GPUs on WPI’s Turing Computing Cluster.

Evaluation

For each of the ten folds trained on the final best-performing model, we evaluate on test data. We calculate final performance metrics directly from the cumulative confusion matrix created by adding together the confusion matrices from each fold. Metrics calculated include precision, recall, and f1-score on each of the five classes as well as overall classification accuracy, as reported in section 3.2.

We additionally calculate confidence intervals of two standard deviations and report these results on each metric using bootstrap sampling as in [7]. We derive intervals from 1000 bootstrapped samples created from the patients’ data by the method described below. One of the twenty patients, patient $x$, is selected at random, and a bootstrapped sample is created from their data. Of the final ten fully-trained models, the bootstrapped sample is fed to the one model where patient $x$ was not in the training or validations set, and a confusion matrix is generated from the output. We repeat this process 1000 times to generate 1000 bootstrapped confusion matrices. Metric averages are then calculated from the cumulative confusion matrix consisting of the sum of the 1000 bootstrapped confusion matrices. Lower and upper bounds of ± two standard deviations are calculated around each metric’s sample mean.
Results

Table 3.1: Cumulative confusion matrix with precision, recall, and F1-Score. Table entries correspond to the number of epochs belonging to a given class as classified by a technician (rows) and by the model (columns). S1, S2, S3, A, R are the class labels for sleep stage 1, sleep stage 2, sleep stage 3, awake, and REM sleep, respectively. Overall accuracy is 81% (see table 3.2 for a more robust bootstrap estimate).

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>A</th>
<th>R</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1299</td>
<td>595</td>
<td>10</td>
<td>382</td>
<td>406</td>
<td>48%</td>
<td>46%</td>
</tr>
<tr>
<td>S2</td>
<td>634</td>
<td>15402</td>
<td>691</td>
<td>438</td>
<td>572</td>
<td>87%</td>
<td>87%</td>
</tr>
<tr>
<td>S3</td>
<td>4</td>
<td>654</td>
<td>4972</td>
<td>71</td>
<td>1</td>
<td>87%</td>
<td>87%</td>
</tr>
<tr>
<td>A</td>
<td>666</td>
<td>178</td>
<td>21</td>
<td>1441</td>
<td>128</td>
<td>59%</td>
<td>57%</td>
</tr>
<tr>
<td>R</td>
<td>373</td>
<td>719</td>
<td>4</td>
<td>260</td>
<td>6348</td>
<td>82%</td>
<td>84%</td>
</tr>
<tr>
<td>Precision</td>
<td>44%</td>
<td>88%</td>
<td>87%</td>
<td>56%</td>
<td>85%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Estimates of metric bounds in the form mean ± 2*stdev. The bottom row reports the mean value of each metric across each class. Overall accuracy is 81.2% ± 0.2%.

<table>
<thead>
<tr>
<th>Sleep Stage</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>45.0% ± 0.6%</td>
<td>47.1% ± 0.5%</td>
<td>43.7% ± 0.4%</td>
</tr>
<tr>
<td>S2</td>
<td>87.5% ± 0.2%</td>
<td>86.4% ± 0.3%</td>
<td>86.7% ± 0.2%</td>
</tr>
<tr>
<td>S3</td>
<td>86.5% ± 0.3%</td>
<td>87.0% ± 0.2%</td>
<td>86.3% ± 0.2%</td>
</tr>
<tr>
<td>A</td>
<td>60.3% ± 0.9%</td>
<td>66.3% ± 0.6%</td>
<td>56.6% ± 0.7%</td>
</tr>
<tr>
<td>R</td>
<td>85.4% ± 0.2%</td>
<td>82.2% ± 0.4%</td>
<td>82.8% ± 0.3%</td>
</tr>
<tr>
<td>Mean</td>
<td>72.9% ± 0.2%</td>
<td>73.8% ± 0.2%</td>
<td>71.2% ± 0.2%</td>
</tr>
</tbody>
</table>

As seen in table 3.1, overall accuracy for the best-performing model is 81%, as calculated from the cumulative confusion matrix. The model attains high individual class performance on Sleep Stages 2, 3, and REM in terms of precision, recall, and f1-scores. The worst performing classes by the same metrics are Sleep Stage 1 and the Awake stage. Table 3.2 shows confidence intervals spanning two standard deviations above and below the means for precision, recall, and f1-score for each of the five sleep stages, as calculated via the method described in section 3.1.4. The average accuracy across all bootstrapped samples is 81%, the same accuracy reported in the cumulative confusion matrix.

Results in table 3.1 and table 3.2 show that our model outperforms prior work [7] on the same dataset, which achieved an overall accuracy of 74% training on one EEG Fpz-Cz signal and a similar breakdown of individual class performance across a shallower,
six-layer model. After modifying our model to train on the same signal used in [7] and evaluating performance, total accuracy decreased from 81% to 80%. This suggests that increasing network depth had a greater effect on performance than adding additional channels.

Comparing our results to human inter-scorer agreement puts some context around the performance we achieve in our project. Whereas one would expect an individual scorer to be self-consistent in her evaluation of sleep stages, a group of scorers may differ in how they score the same epochs. These rates of inter-scorer agreement would then serve as performance goals for a model trained on many scorers classifications, such as ours.

Table 3.3: Inter-scorer agreement conducted by the AASM: reproduced from [31] with proxy precision, proxy recall, and accuracy reported. Overall inter-scorer agreement is 82%.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>A</th>
<th>R</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>187,634</td>
<td>64,700</td>
<td>205</td>
<td>32,545</td>
<td>12,910</td>
<td>63%</td>
</tr>
<tr>
<td>S2</td>
<td>116,274</td>
<td>1,570,861</td>
<td>121,141</td>
<td>13,080</td>
<td>23,180</td>
<td>85%</td>
</tr>
<tr>
<td>S3</td>
<td>298</td>
<td>87,033</td>
<td>181,337</td>
<td>224</td>
<td>350</td>
<td>67%</td>
</tr>
<tr>
<td>A</td>
<td>29,658</td>
<td>11,222</td>
<td>779</td>
<td>250,434</td>
<td>5703</td>
<td>84%</td>
</tr>
<tr>
<td>R</td>
<td>25,875</td>
<td>22,883</td>
<td>342</td>
<td>6624</td>
<td>531,611</td>
<td>91%</td>
</tr>
<tr>
<td>Prec</td>
<td>52%</td>
<td>89%</td>
<td>60%</td>
<td>83%</td>
<td>93%</td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>57%</td>
<td>87%</td>
<td>63%</td>
<td>86%</td>
<td>92%</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3 shows results reproduced from [31], a study that compiled more than 2,500 human scorers’ classifications on 1,800 epochs for a total over 3,000,000 scoring decisions. Table 3.3 can be thought of as a confusion matrix, the rows representing ground truth for each sleep epoch (interpreted as majority score attributed to the epoch) and columns as the classifications of the collective pool of scorers. Overall inter-scorer accuracy is 82%, which is on par with our results. Comparing with table 3.1 and table 3.2, human inter-scorer agreement performs better than our deep CNN on Stages S1, Awake, and REM, while it matches our network’s accuracy on Stage S2; and our network’s accuracy on Stage S3 exceeds inter-scorer agreement.

We hypothesize that these differences in recall and precision between our model and [31] arise from the large variance in human inter-scorer agreement and the small number of scorers who scored our dataset. High variation between scorers is supported by other research in inter-scorer accuracy; reviews of inter-scorer agreement and meta studies report similar or worse results for overall inter-scorer accuracy and varying performance across classes, with the highest accuracy from stage REM (78-94%) and sleep stage 2 epochs (79-90%) and the lowest from stage 3 (69%) [22, 32]. Surprisingly, intra-scorer agreement appears to be comparable to inter-scorer agreement; one study reports accuracies of the same scorer on data re-scored after a median of 6.5 months to be in the range of 79-87% [32].

Since any machine learning model can only aspire to learn as well as the original classifier of the samples, the maximum reported inter-scorer reliability of 82% serves
as a general bench for comparing model performance. Our model's accuracy of 81% is on par with such a benchmark.

Recent results reported on the same dataset and published at the time of writing [33], further suggests that our model is close to extracting as much sleep stage information from EEG data as human experts. [33] reports achieving a maximum accuracy of 82%, only a 1% improvement over our results, using a more complicated CNN and bidirectional LSTM model for sleep classification. We note that their results mirror our observations and the inter-scorer literature review that suggest human performance flat-lines around 80% accuracy.

**Discussion**

In the first phase of our project, we explore the use of deep convolutional neural networks for classification of sleep stages derived from multi-channel polysomnogram data. Classification accuracy of the proposed technique reaches 81%, surpassing that of prior work [7] that uses a single-channel version of the same data set. Reducing the number of input signal channels from three to one brought accuracy down to 80%, thus suggesting that the performance gains of our deeper, multi-channel model over the prior research in [7] derive mainly from increasing network depth.

Currently, neural networks are able to achieve performance on par with a committee of human experts; our network approaches this level of performance, attaining a classification accuracy that is comparable with human inter-scorer agreement. Because sleep stage labels are human constructs following human-defined rules, we believe that further improvements in sleep stage prediction will only occur with changes outside of the computer scientist's control, such as scorers' behavior becoming more consistent via better training or changing categorization rules. It may not be fruitful to pursue substantial further improvements to sleep scoring performance in future work in the absence of improvements to the staging standards themselves.
Chapter 4

Sleep Signal Feature Extraction

One of the clear disadvantages of using neural networks for any supervised classification problem is that their decision-making logic is not discernible. This problem arises from the inscrutability of the stacked series of non-linear functions applied to the features of the data; these complex functions can individually be thought of as maps sending preceding feature inputs into a new feature space. It is therefore ironic that there exists some correlation between model complexity, and thus inscrutability, with higher levels of performance, as we demonstrate in section 3.2 of chapter 3. This combination of the black-box nature of the feature maps in neural networks with their exceptional performance leads to new research questions on how to better understand the mechanics of neural networks and how to utilize the features they create.

In phase two of this project, we attempt to exploit this ability of neural networks to project data into new feature spaces. Our goal is to explore a specific conjecture about sleep signal data and its classification into sleep stages; we claim that their may exist features in sleep data unknown to sleep technicians that discriminate between sleep signals, and that there may be classes that could better demarcate the data. If these new features exist and become known, they could aid in creating new classification rules that may lead to hypnograms that are more consistent across technicians, a problem discussed in section 3.2. If new classes exist and become known, their discovery could lead to a shift in what categories of sleep are recognized and used by sleep experts, which may further improve consistency or lead to other discoveries. As a simple example, if there were to exist two types of S2 sleep and we could discern between the two, it is conceivable that we could see correlations between the new classes and illnesses that were previously unobservable due to the prior composition of the two classes.

To explore this conjecture, we experiment with features generated by autoencoders and attempt to cluster the projected data their encoders create. If we observe distinct clusters that do not align with the current paradigm’s class labels, the results could suggest that existence of a different set or demarcation of labels that may more
"naturally" describe the data. Chapter 4 discusses the methodology and experiments we conduct. In the process of building our networks, we create alternative methods of training autoencoders and compare our approaches to current practices. We additionally experiment with training supervised models on the outputs of encoders to test if the constructed features exhibit better classification properties than supervised networks trained with raw data.

Methodology

Our methodology for feature and cluster discovery involves first training autoencoders for the purpose of projecting EEG signal data into new feature spaces and then applying unsupervised clustering and supervised learning methods to projected data. Autoencoder architectures, training procedure experiments, and data exploration approaches are described in the following sections.

Autoencoder Architectures

The inputs to our autoencoder models consist of the same polysomnogram data sourced from the Study 1 data of the Sleep-EDF Database [Expanded] [24], a subset of the publicly-available Physionet Database [23]. We reuse a fold generated from the final sleep-stage classification training experiments; the test set, validation set, and training set contained 2 patients, 3 patients, and 15 patients, respectively, where each patient’s data is composed of two night of signal recordings. During the course of experimentation, we use the training set for training encoders, the validation set for tuning the hyperparameters of both the networks and the clustering methods, and the test set for reporting results.

Our earlier findings reported in chapter 3 show that, of the three signals that comprise the sleep patients’ polysomnograms, EEG data from electrodes placed in the Fpz-Cz positions contributed considerably to classification accuracy, whereas the remaining signals did not. Taking advantage of this observation, we only use one channel of EEG Fpz-Cz signal for training autoencoders. Therefore, raw data for training autoencoders has dimensions (15000,1), comprised of 150-seconds, or 5-epochs, of EEG data sampled at 100 Hz. It is important to acknowledge that, in choosing to work with this signal, we are implicitly presuming that the most discerning data used for supervised classification would be the best data for uncovering unknown features or clusters. This is not necessarily the case; an argument can be made that searching for features in the less-discerning signals is more productive if the supervised network is ignoring hidden features there in favor of easier-to-detect correlations. Future work could focus on incorporating these signals into autoencoder training procedures or focusing on them exclusively.

We design and train two classes of autoencoders for use during the experimentation and clustering phases of the project. The first variety takes the raw signal data from the patients as an input. We construct two variations: one which uses the full (15000,1) dimension as an input, as in chapter 3 and [7], and one which uses only one epoch of sleep data, equivalent to inputs of size (3000,1). Features are scaled to be between
zero and one. Sleep stage classifications are often based on the frequencies of signal observed in polysomnogram data \[11\], and so we train a second class of autoencoders aimed at uncovering similar relationships by projecting the original signals into the frequency domain using the Fast Fourier Transform. We use data composed of 5 sleep-epochs and 1 sleep-epoch, which result in frequency spectra and inputs to our encoders of shape (7500,1) and (1500,1), respectively. Features are scaled to be between zero and one.

As in chapter 3, we structure the autoencoders as convolutional neural networks. The networks contain pooling layers for creating bottlenecks in the encoder and up-sampling layers for increasing dimensionality in decoders. An argument can be made that using convolutional neural networks on the frequency spectra of PSG data is not appropriate, as any correlations between features in this space are not spatially-dependent. However, relationships could exist between disparate features which justifies the use of fully-connected layers. However due to the size the feature space, training fully connected autoencoders is impractical with our resources, and we choose to use convolutional layers to reduce model complexity.

The final architecture implemented after experimentation is described below. Encoders are composed of encoding and decoding stages, which are comprised of stacked convolutional layers sandwiching either a max pooling layer that down samples its input by a factor of two or an up-sampling layer that doubles the size of its input. We train encoders and decoders with varying numbers of up-sampling and down-sampling stages ranging from one to six. To ensure that rounding errors do not lead to mismatched layer dimensions between encoders and decoders, the inputs to the encoders are padded with features set to zero to make the dimensionality of the input layer divisible by \(2^6\). The number of convolutional layers is constant across all stages in the autoencoder, with the exception of the first stage of the encoder and last stage of the decoder, which have 1.5x more convolutional layers. A visualization of a three-stage autoencoder with two convolutional layers per stage accepting data inputs of dimension (15000,1) can be seen in fig. 4.1.

Hidden layers of the networks have Leaky Relu activations \((\alpha = 0.3)\) \[34\] to address problems encountered with ReLus and unstable training performance. As discussed above, input layers are scaled to be between zero and one. A final convolutional layer is added to the end of the encoder with a sigmoid output to similarly have outputs between zero and one.
Figure 4.1: Example of an autoencoder model. There are three stages in the model with two convolutional layer per stage. Note that the first encoder and last decoder have additional convolutional layers.

Autoencoder Training

Layer-Wise Training

Section 2.5 describes the layer-wise training procedure used for training autoencoders, and ?? provides a pseudo-code representation of this method. In our project, rather than a strictly layer-wise approach, we train autoencoders in a stage-wise fashion. Two stages, a down-sampling encoder stage and an up-sampling decoder stage as described in section 4.1.1 make up each shallow autoencoder. Larger final autoencoders are then assembled from the pretrained shallow autoencoders’ stages and trained on the original data. In this project, we will use the phrase stage n shallow autoencoder to refer to the small autoencoder that trains on data that has been down-sampled n times, and the phrase stage n autoencoder to refer to the full-size autoencoder created from the stage 1–n shallow autoencoders. Thus a stage n autoencoder’s narrowest layer
will output \( \frac{1}{r} \) as many features as the original input, ignoring the effects of padding. An example of how shallow autoencoders and their stages stack to construct full-size autoencoders is illustrated in fig. 4.2 on a stage 3 autoencoder. We train each shallow autoencoder using RMSprop [35] with a learning rate of 0.001 and \( \rho = 0.9 \). Stage 1 shallow autoencoders train to minimize binary cross-entropy loss whereas deeper shallow autoencoders train to minimize mean squared error.

Figure 4.2: Example of smaller pre-trained autoencoders used to initialize a final, full-size model. There are three stages in the model with two convolutional layers per stage. Note that the first encoder and last decoder have additional convolutional layers.

**Bow Tie Training**

In addition to training with the layer-wise method above, we introduce a new method inspired by [36] for training autoencoders that we theorize could offer certain advantages over the current layer-wise training practice. [36] demonstrates that neural networks for supervised learning train faster and reach higher levels of accuracy when trained with the following iterative deepening procedure. First, a shallow network is initialized to have an input layer that accepts the original data and an output layer that outputs the class predictions. In each successive deepening, a hidden layer is inserted before the output layer, and the new hidden layer’s weights are set such that the layer acts as an identity function. Finally, the new deepened network is retrained. With this method, each new deepened network applies the same function prior to training as its fully-trained predecessor.

26
The consensus of why supervised networks trained in this fashion perform better than a full network trained in an end-to-end fashion is that the weights of smaller networks serve as good initializations for more complex networks trying to do the same task. This is very similar to the observations and logic used to justify training autoencoders in a layer-wise fashion. However, unlike unsupervised layer-wise training for autoencoders, in [36] only one network trains rather than multiple shallow networks. Applying this new supervised training method to training autoencoders could lead to certain advantages over the current layer-wise training method for use in autoencoders, which we discuss below following a description of our adaptation of the method in [36].

In our variant, an initial stage 1 shallow autoencoder trains on the dataset. For each successive step \( i \), a new stage \( i + 1 \) shallow autoencoder is inserted into the center of the prior autoencoder, and the entire network re-trains. Thus, with each deepening, one autoencoder grows until it reaches the desired depth. Because our network down-samples and up-samples the data, we choose to initialize hidden layer weights randomly. We refer to this method of training as **bow tie training**, named for the visual resemblance of a tightening bow tie to the growing autoencoder. We refer to an autoencoder trained with \( n \) decoder and encoder stages as an **stage \( n \) bow tie autoencoder**. Figure 4.3 illustrates an example of how a stage 3 bow tie autoencoder trains. We train each bow tie autoencoder using RMSprop with a learning rate of 0.001 and \( \rho = 0.9 \) to minimize binary cross-entropy loss.

![Figure 4.3: Example of iteratively-deepened training procedure. There are three stages in the model with two convolutional layers per stage. Note that the first encoder and last decoder have additional convolutional layers](image-url)
Training Comparisons

We claim that bow tie training may be a better choice for training autoencoders due to its simpler implementation and potential for improving accuracy. Unlike the layer-wise training methods, bow tie training maintains the same input and output layers during all sub-training procedures. Therefore data does not need to be down-sampled prior to training; because each new shallow autoencoder gets embedded within the current model, the growing autoencoder takes care of ”preprocessing” the data through the encoder stages. This reduces implementation complexity as a separate data preprocessing function does not need to be coded for training the stage $n > 1$ shallow encoders. We believe that there may be improvements to accuracy due to certain artifacts of layer-wise training. As neural networks generally contain more than one activation function, layer-wise training could involve using multiple loss functions. In our example, the input and output to the autoencoders are vectors of values ranging between zero and one and binary cross-entropy is minimized. However, internal layers comprise of ReLU-like activation units have the potential to take on values between zero and infinity, making mean squared error loss a more appropriate choice for the model. It is possible that this miss-match of loss functions between the final training procedures and the shallow autoencoder training procedures could lead to weight initializations farther from local minima than could be achieved through other methods. Changing to activation functions that are the same output as the final layer could mitigate this problem, but only at the expense of losing the power of using ReLU-based activations and introducing vanishing gradient issues with sigmoid or tanh activations.

We also claim that bow tie training could lead to faster training for ensembles of autoencoders. Training each shallow autoencoder with the layer-wise approach should be relatively faster than training the successively-deepened bow tie network. However, as each stage $i$ bow tie autoencoder finishes training, copies of the model can be saved, and thus a running set of autoencoders can be generated as a consequence of the training procedure. However, after training shallow autoencoders with the layer-wise approach, each full size stage 1–$n$ autoencoder needs to train separately in order to generate the set of autoencoders. Therefore though it may be faster to use layer-by-layer training to generate certain autoencoder of depth $n$, if an entire set of autoencoders of varying depths is desired and the training procedures lead equivalent performances, bow tie training may be faster and better option.

From implementing both training methods, we note that bow tie training is simpler to implement. In section 4.2.1 we evaluate the performance and speed of each method and compare results from training runs on patients’ sleep signal data.

Clustering Experiments

Following the training of autoencoder models, we attempt to search for previously-unknown clusters in data by first passing data through the trained encoders and then applying visualizations and unsupervised learning methods to the encoded data. As discussed in section 4.1.1, we train models on both raw input data and data mapped to the frequency domain. We then perform clustering experiments on the encoded raw and encoded spectrogram data.
For visualizing the data, we use Multidimensional Scaling (MDS) to project encoded data into a two-dimensional space, and we compare results with the alternative t-SNE projection technique [37]. For quantitative cluster discovery, we use a variety of unsupervised learning techniques described below. Inspired by work in [38] which successfully used the Chinese Whisperer graph clustering algorithm [39] to discover new varieties of dolphin clicks in raw signal data, we use the same algorithm as well as variants of spectral clustering as described in [40] to uncover new clusters in sleep data. We also attempt to find clusters using DBSCAN [41] and using the Expectation-Maximization algorithm [42] in combination with different assumptions made on probability distributions of the encoded features.

In section 4.2.2 we discuss the results from visualizations made of encoded data and the performance of our clustering methods.

Supervised Learning Experiments

In order to compare the relative quality of features created from different depth encoders, we evaluate the performance of the encoded features at classifying sleep signals into sleep stages and compare the results. As each encoder trained contains different numbers output features depending on both depth and the dimension of the input signal/spectrogram, we choose to use a classification model that is simple and fast to train across many different data sets. Our experiments use decision trees to evaluate clustering performance of the new feature spaces. Because spectrogram data does not have a temporal component that is translation invariant, it is also possible to train decision trees on the input spectrogram data as well. Thus, for spectrogram data, we compare supervised classification performance between encoders and the original input data. In section 4.2.3 we discuss our findings.

Results

Autoencoder Training Experiments

We evaluate the performance of layer-wise training and bow tie training by comparing results and times obtained from applying each procedure to training the same model architectures. Experiments are conducted on WPI’s Turing computer cluster running on the NVIDIA cuda platform. Each procedure trains a model using the same data-loading functions and single K20 GPU.

Timing Experiments

We conduct timing procedures for both the methods as follows. After loading packages and validation data, time starts at zero. For layer-wise training, we record the training times of each shallow autoencoder from stage 1 to n, followed by the training times of each successive full-size autoencoder from stage n to 1. For bow tie training, we time the training of each successively-deepened full-size autoencoder from stage 1 to n.
Although we do not expect to see times that are different across model configurations, we nonetheless run multiple experiments with different model architectures for comparing results. The two tunable parameters we vary to test timing are the number of stages in the model and the number of convolutional layers per stage. For the layer-wise method, the number of epochs refers to the epochs used for training each shallow and each fully-assembled autoencoder. For bow tie training, the number of epochs refers the epochs used for training each successively-deepened autoencoder. Because the primary goal of these experiments is to measure and compare training times, we train for short durations of time without stopping criteria. We limit our experiments to runs where the number of stages vary between two and six and the number of epochs varies between two and six.

Table 4.1 shows the training times observed during the layer-wise training experiments and Table 4.2 shows the training times observed during the bow tie training experiments. Looking at Table 4.1 we see that training time for each shallow autoencoder decreases with depth, which is expected as each shallow autoencoder has fewer weights than its predecessors. Looking at Table 4.2, we see that training time for each autoencoder increases with depth, which is also expected because the autoencoder is iteratively being deepened and thus acquiring more trainable weights. When comparing the two tables, layer-wise training exhibits faster training times for training a single, full-sized autoencoder. However, bow tie network training, as an artifact of the training procedure, generates an ensemble of autoencoders faster than layer-wise training. In experiments using fewer than six stages, we come across a threshold where an ensemble of bow tie encoders can train faster than even a single full-sized model trained in layer-wise fashion. Thus, depending on model architecture, we posit that there exists a tipping point where the added time required to preprocess data through shallow encoders outweighs the time to iteratively train bow tie encoders. In conclusion, our experiments suggest that bow tie training is at least faster than layer-wise training for creating ensembles of autoencoders, and in certain circumstances can be faster than layer-wise training generally.

Reconstruction Experiments

Though bow tie training exhibits certain timing advantages over layer-wise training, these advantages are only meaningful if bow tie training produces autoencoders whose reconstruction loss is on par with or better than autoencoders trained in a layer-wise fashion. To test the reconstruction performance of the training methods, we train models as described above for a larger number of epochs, ranging from 100 to 1000, and compare reconstruction loss across the ensembles of full-size autoencoders generated from each procedure.
Table 4.1: Layer-Wise Training Time Experiments. Times listed in seconds. The "Layers" column refers to the number of convolutional layers in each stage. Table lists the amount of time for training each shallow encoder, numbered 1–6, the final training time of the deepest, full-size autoencoder, and the time required to train the ensemble of all full-sized autoencoders.

<table>
<thead>
<tr>
<th>Epochs</th>
<th>Layers</th>
<th>Shallow Model Training Times</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 2 3 4 5 6</td>
<td>Deepest</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>192 98 83 75 76 73</td>
<td>893</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>306 148 121 108 103 100</td>
<td>1395</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>423 198 158 140 132 130</td>
<td>1902</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>354 188 159 148 144 144</td>
<td>1677</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>583 289 236 213 203 201</td>
<td>2659</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>812 382 310 274 259 255</td>
<td>3621</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>517 276 235 222 214 213</td>
<td>2459</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>857 423 348 315 301 299</td>
<td>3900</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1205 571 456 405 382 373</td>
<td>5322</td>
</tr>
</tbody>
</table>

Table 4.2: Bow Tie Training Time Experiments. Times listed in seconds. The "Layers" column refers to the number of convolutional layers in each stage. Table lists the amount of time for training each shallow encoder, numbered 1–6, the final time to train the deepest model, and the total time required to train the ensemble of all full-sized autoencoders.

<table>
<thead>
<tr>
<th>Epochs</th>
<th>Layers</th>
<th>Bow Tie Model Training Times</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 2 3 4 5 6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>192 230 260 283 305 329</td>
<td>1599</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>307 382 438 482 523 572</td>
<td>2704</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>422 533 613 681 748 810</td>
<td>3807</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>356 435 486 520 548 576</td>
<td>2921</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>583 737 830 893 945 998</td>
<td>4986</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>828 1036 1173 1263 1342 1431</td>
<td>7073</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>518 638 708 752 782 816</td>
<td>4214</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>874 1084 1213 1298 1361 1423</td>
<td>7253</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1206 1532 1724 1843 1938 2031</td>
<td>10274</td>
</tr>
</tbody>
</table>
Figure 4.4: Reconstructed EEG Signal Outputs for a Stage 5 and Stage 6 Bow Tie autoencoder on data from the Validation Set. The model contains one convolutional layer per stages and used 100 epochs of training per stage. The graph demonstrates the differences in reconstruction accuracy—results that on the surface do not intuitively follow from differences in average reconstruction loss.

Figure 4.5: Validation loss during training of Stage 4–6 Bow Tie Encoders. Down sampling during each deepening reduces bottleneck dimensionality by a factor of two, and loss values increase by on an order of \(10^{-3}\).
In all training runs and comparisons between training methods, the magnitude of the difference between the two training procedures’ loss values is quite small—on the order of $10^{-3}$. On the surface, we would expect differences of this magnitude to be meaningless. However, in practice, we find that models with average loss values that differ by such an order of magnitude show observable differences in signal reconstruction performance. Figure 4.4 illustrates these differences in reconstruction error and fig. 4.5 illustrates the differences in loss values between models trained with different numbers of stages. We attribute the small differences in loss values to the generally-good performance of all the models and that most of the reconstructed features in our data fall within a small range of their original values.

We compare reconstruction performance by measuring the average loss obtained from bow tie and layer-wise training as follows. For models trained with $n$ stages, we compare the average validation loss values achieved on the stage $n^{th}$ bow-tie model to the final stage $n$ layer-wise model trained on re-assembling the shallow autoencoders. For all model depths and number of convolutional layers, and for a sufficient minimal number of training epochs, we find that bow tie-trained models and layer-wise-trained models perform equivalently, reaching the same level of loss well within an order of magnitude of $10^{-3}$. Layer-wise models consistently start with lower initial loss values, and bow tie models’ loss values quickly converges to the layer-wise values. We notice that bow tie validation loss exhibits spikes during training not present in layer-wise validation losses. This occurs at all stages of deepening. We do not have a theory as to why the volatility occurs during bow tie training and not layer-wise training. However, as it is possible to save the best-performing models, we do not consider the volatility a significant problem. Section 4.2.1 illustrates three examples of trained models that reflect the observations above.

**Final Comparisons**

From our experiments training autoencoders using both procedures, we find bow tie training to be equivalent to layer-wise training, each offering certain training time advantages, depending on the desired output. In our project, we find generating ensembles of autoencoders to be more useful than returning single autoencoders, as we test each for their unsupervised clustering and supervised classification abilities. We therefore use bow tie training for all the following experiments in our project.
Figure 4.6: Examples of validation loss during the training of full-sized autoencoder. All graphs demonstrate the presence of spiking in bow tie validation loss and the equivalent levels of performance between layer-wise and bow-tie training. Top: model with stages containing one convolutional layer trained for 100 epochs. Middle: same model as above, trained with plateau stopping criteria. Included for comparison is validation loss from training a full-size with random initializations. Bottom: model with stages containing two convolutional layers trained for 250 epochs.
Encoder Clustering Experiments

We assess the clustering performance of our encoder models through qualitative visualization techniques and quantitative approaches to measuring cluster differentiability. As discussed section 4.1.1, we train autoencoders on different types of input data: two models trained on raw input data and two models trained on Fourier-transformed data, mapping the original signals into the frequency domain. Raw data inputs are either one 30-second sleep epoch sampled at 100Hz, equivalent to an input dimension of 3000, or five 30-second sleep epochs sampled at 100Hz, equivalent to an input dimension of 15000. Input data mapped into the frequency domain similarly comes from either one sleep epoch of raw data, leading to an input dimension of 1500, or five sleep epochs of raw data, leading to an input dimension of 7500. Following the observations made in section 4.2.1, we train the four model variants above using bow tie training to generate ensembles of encoders. We find that the reconstruction loss plateaus after around 50 epochs of training per deepening iteration, regardless of the model input or number of convolutional layers per stage. We report results on models trained with 100 epochs per deepening, 4 convolutional layers per stage, and containing up to six stages of deepening. These examples reflect our general findings training with other architecture and parameter settings.

Cluster Visualizations

In order to qualitatively assess the clustering ability of the encoders, we use Multidimensional Scaling (MDS) to project encoded data into a two-dimensional space, allowing for simple visualization; for each of the four autoencoder variants described above, data is passed through the encoder and visualized using this technique. With both raw and spectral data, we notice that encoded data appears to form clusters that correlate with the original class labels, as illustrated in fig. 4.7. Encoded raw data appears to fall in concentric rings around a central point, with different classes appearing to fall with high frequency at different distances away from the origin. We theorize that this projection may result from high-dimensional encoded data falling into different concentric high-dimensional balls in the encoded space. Spectrogram data, when passed through an encoder and projected into two dimensions, does not exhibit the same concentric circle pattern and rotational symmetry as the raw encoded data projections. We observed these patterns across all encoder depths and input data sizes; all experiments suggest that visually, the greatest change in cluster formation and differentiation occur when changing between raw EEG signal inputs and spectrogram inputs. Results do not differ from those generated using the alternative t-SNE projection technique [37].

Across both varieties of data, the MDS projections exhibit similar differentiation abilities between class types, with perhaps encoded spectrogram data exhibiting clearer boundaries between clusters. None of the encoders appear to project classes of data into more than one distinct cluster. Interestingly, projected data loosely coincides with the performance of the supervised classification network described in section 4.1.1 whose confusion matrix is reported in table 3.1. Looking at the encoded spectrogram data in fig. 4.7 and comparing to supervised learning results, we notice that the distinctness of the S2 and S3 clusters mirror the high precision and recall achieved in supervised classification while, the disparate spread of S1 and Awake data points
mirror the low precision and recall achieved in supervised classification. These observations along with the clear differentiation visible from MDS projections suggest that data passed through encoders retain relevant features that differentiate classes from each other.

Processing EEG signals through a convolutional encoder projects the original data into a space where the data loses its temporal quality. This allows for the ability to cluster the projected data and see meaningful patterns; clustering raw signal data, however, would result in nonsensical visualizations, as the visualizations would fail to account for the temporal symmetry present in data of the same class. Because mapping signal data into spectrograms also strips data of its temporal meaning, it’s possible to not only cluster encoder outputs, but also encoder inputs and compare between the two. We compare MDS visualizations for spectrogram data inputs and encoder outputs and notice that both cluster data in a similar fashion, as illustrated in fig. 4.8.

In summary, our visualization experiments suggest that the encoder preserve features from the original data that differentiate between classes. There is no suggestions from these experiments that encoders created new clusters that were previously-unknown to researchers.
Figure 4.7: MDS projections of encoded test set data. Figure 4.7a: inputs are five sleep epoch of raw EEG signal data fed to a stage 6 encoder. The encoder maps the input data of dimension 15000 into encoded data of dimension 236. Sleep Stage S2 is left out for image clarity. Figure 4.7b: inputs are one sleep epoch of EEG spectrogram data fed to a stage 6 encoder. The encoder maps the input data of dimension 1500 into encoded data of dimension 24.
Figure 4.8: MDS projections of spectrogram data and encoder output. **Figure 4.8a:** projection of spectrogram data. Sleep Stage S2 is left out for image clarity. **Figure 4.8b:** inputs are five sleep epochs of spectrogram data fed to a stage 3 encoder. The encoder maps the input data of dimension 7500 into encoded data of dimension 944. Sleep Stage S2 is left out for image clarity.
Cluster Discovery

In addition to visualizations, we attempt to quantitatively discover clusters in data using standard unsupervised approaches. We found that the most promising results come from encoders trained on spectrogram data. However, our experiments did not yield results that add additional information beyond what we observe with earlier MDS projections.

Our approach is to apply clustering procedures to encoded data, score the differentiation quality of the clusters, and select clustering models that achieve the high measurable differentiation between the clusters found. These selected models could then represent a more “natural” clustering paradigm than the human-created sleep stages currently used by technicians. As discussed earlier, finding clusters that do not correlate with the standard sleep stages could lead to groupings useful for data inference. Two examples of clustering approaches we evaluate that follow this method are described below. In one set of tests, we assume features in the encoded space come from a Gaussian mixture model, use the Expectation-Maximization algorithm to find clusters, and evaluate performance using Bayes Information Criteria. In a second set of tests, we use spectral clustering algorithms with different graph Laplacians as described in [40] to cluster data and evaluate clustering differentiation by using a scoring criterion such as Silhouette Coefficients [43].

For the majority of the clustering experiments, models did not detect the presence of naturally-occurring clusters in the data. This was made evident by the often decreasing cluster scoring criteria with the addition of more clusters to the models; thus, often the best-scoring models were ones that only detected a single cluster. This was especially evident with raw encoded data. Spectrogram data performed slightly better when using spectral clustering; the best-scoring models contain three or four clusters that appear to loosely correlate with the observed clusters visualized with MDS. However, these results only confirm the observation that the encoders preserve features from the original data that differentiate between classes and not reveal additional information about new clusters present in encoded data; none of the clusters found through unsupervised methods pointed toward groupings that were not previously known. In fact, clusters generally combined classes into larger clusters and were not able to differentiate between distinct already-known classes. As with MDS, cluster homogeneity correlate with the performance of supervised learning as described in Chapter 3; the most homogeneous clusters found contained data from the classes S2 or S3: the same classes that performed the best after supervised learning.

In summary, our clustering experiments confirm the observations we make from our visualizations: our encoders preserve features from the original data that differentiate between classes, but do not appear to expose or create new clusters that were previously-unknown to researchers.

Supervised Learning Experiments

We compare the relative quality of features created from different depth encoders by evaluating the performance of encoded features at classifying the sleep stages. Our goal is to see whether encoders can produce features markedly different from the original input data. If encoders simply act as lossy compressor, one would expect classification
Table 4.3: Stage 1–6 Signal and Spectrogram Decision Tree Performance. Encoders trained with 100 epoch per stage deepening and contain 4 convolutional layers per stage. Encoder inputs contain 5 epochs of sleep stage data.

<table>
<thead>
<tr>
<th>Data Stage</th>
<th>Raw Signal Decision Tree</th>
<th>Spectrogram Decision Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data Dimension</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Input Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stage 1</td>
<td>7552</td>
<td>39%</td>
</tr>
<tr>
<td>Stage 2</td>
<td>3776</td>
<td>41%</td>
</tr>
<tr>
<td>Stage 3</td>
<td>1888</td>
<td>39%</td>
</tr>
<tr>
<td>Stage 4</td>
<td>944</td>
<td>41%</td>
</tr>
<tr>
<td>Stage 5</td>
<td>472</td>
<td>46%</td>
</tr>
<tr>
<td>Stage 6</td>
<td>236</td>
<td>40%</td>
</tr>
</tbody>
</table>

As seen in table 4.3, with the exception of the stage 5 model, performance of raw signal decision trees does not appear change as the depth of encoders increases, suggesting that features for classifying signal data persist after passing through encoders. In the stage 5 model, we observe accuracy increasing from 40% to 46%. When testing other models with different numbers training depths, convolution layers per stage, and input dimension sizes, we find similar levels of performances as the decision trees in table 4.3 with increasing accuracy for stage 4–5 raw signal decision tree models.
We theorize that the increased accuracy for stage 5 decision trees on raw data and for all stage decision trees for spectrogram data arises from the regularization effect of using encoders. Similar to pooling layers described in section 2.3, encoders down sample data, which leads to regularization of a classifier built on encoded features, as there are fewer features, or degrees of freedom, for a classifier to work with. Single decision trees in general are known to over-fit data. Thus, it is not surprising that encoded data could perform better at classification tasks than the original data if trained with single decision trees. Indeed, our theory is validated by the high performance on training data; each decision tree has accuracy above 90%. Therefore, the encoders are likely behaving as model regularizers rather than directly creating more separable features than what already exist in the original data.

**Discussion**

Our autoencoder bow tie training algorithm achieves the same level of reconstruction accuracy as layer-wise training. Although training a single autoencoder requires less time using layer-wise training, because the bow tie algorithm trains successively deepened full-sized autoencoders, bow tie training creates ensembles of encoders faster than layer-wise training. This is particularly useful for experiments that involve the evaluation of different depth encoders, such as the supervised experiments we conduct in section 4.2.3. We therefore conclude that bow tie training is an effective alternative procedure for training autoencoders. Future research could focus on understanding why validation loss appears to be more volatile than in layer-wise training, as described in section 4.2.1.

Our clustering results do not achieve the desired goal of discovering, or pointing to the existence of, new clusters in sleep signal data. Future work could focus on simple modifications to our current approach. As discussed in section 4.2.1, in choosing to work with the best-performing EEG signal used in the supervised classification task, we presume that the most discerning data used for supervised classification would be the best data for uncovering unknown clusters. However, searching for features in the less-discerning signals may be more productive because the supervised network could be ignoring hidden features in favor of easier-to-detect correlations. Thus, using the less prominent signals may have a higher chance of leading to unexpected results. There is also potential in focusing on a single sleep stage; rather than training on all the signals, focusing on a single stage that according to experts would benefit from subdivision may lead to discovery of class sub-types that would aid in sleep stage classification.

Our supervised classification and visualization results show that encoders can create feature spaces that retain useful information about sleep signal data. A thorough treatment of encoder’s performance could focus on comparing classification between data projected into encoded spaces versus data projected into other spaces created by known down-sampling algorithms, such as Principal Component Analysis. Exploring what comprises the encoded spaces could be a direction for future research. By perturbing the new features and using decoders to reconstruct the signals, we could create modified signals in the original features whose visual patterns may reveal information.
on what the down-sampled features encode. Sharing these findings with experts in the sleep domain may lead to insights and future research directions.
Chapter 5

Conclusions and Future Work

Sleep stage classification is an important task in clinical medicine in which human experts map EEG and other time-varying sleep signal data to a sequence of standard sleep stages known as a hypnogram. In the first phase of our project, we successfully build and train a deep convolutional neural network that classifies PSG signal data into sleep stages with an accuracy on par with human inter-scorer agreement. We believe that further advancements in sleep stage classification performance can only be achieved if improvements to human accuracy increase with changes to AASM scoring procedures, better training, or a change in scoring paradigm.

In the second phase of our project, we train deep convolutional autoencoders and use the respective encoders for projecting sleep signal data into new feature spaces. In conducting our exploratory work on encoded spaces, we implement a new iterative autoencoder training technique of our own design called bow tie training. We discover that bow tie training performs as well as layer-wise training and trains ensembles of autoencoders is less time. Future research could focus on understanding why validation loss appears to be more volatile than in layer-wise training. We apply unsupervised clustering methods to the projected data with the goal of discovering clusters in signal data that do not correspond to already-established sleep stage classes. Although we do not find new clusters within the encoded feature spaces, through our visualization and supervised learning experiments, we show that encoded spaces preserve information about class labels, particularly in the case of Fourier-transformed raw signal data. Future work could focus on further exploring the relationship between the original feature space and encoded spaces; experiments involving perturbing features in the encoded space and reconstructing the signals could be conducted to explore the map from raw data to encoded spaces. Sharing these findings with experts in the sleep domain may lead to insights and future research directions.
Bibliography


