A Deep Learning Model for Automated Sleep Stages Classification Using Single-Channel PSG Signals

Martin Kjäer, Lukas Kulbis, Rahul Shah

May 2019

Abstract

Sleep, an essential part of life, is a grossly misunderstood and mysterious topic in the field of science. To date, sleep staging is the most prominent method of quantitatively measuring the state of the resting mind; however, this crucial task involves the analysis of multiple signals, thus making it very tedious and complex. Trained experts spend hours manually annotating polysomnogram (PSG) signals from a single patient, but could use this time more productively by analyzing a patient's sleep patterns to gain valuable insights. In this work, we design a deep learning network that can self-learn the relevant features from PSGs, specifically the EEG and EOG channels, in order to accurately classify different stages of sleep using two public databases (sleep-edf and sleep-edfx). Using a 1-D CNN with 19 layers, we were able to create reproducible results with 78% accuracy when classifying into 6 sleep classes and 97% when classifying into 2 classes.

1 Introduction

Sleep is an essential part of life, yet very little is known about it. Lack of sleep often results in negligible annoyances such as drowsiness or lack of focus, but can lead to serious health problems in certain individuals [Panossian & Avidan, 2009]. Recent research evidence indicates that modulating sleep activity patterns, via peripheral stimulation at specific sleep stages can be beneficial in a wide range of contexts including memory acquisition, memory consolidation [Marshall et al., 2004, 2006] and relief from depression [Vogel et al., 1980]. By designing a deep learning network which provides insight by automatically classifying different sleep stages, sleep experts can spend more time analyzing patterns and provide valuable insights for their patients.

With the emergence of affordable consumer sleep technologies, there is an opportunity to study sleep on large population groups. As most of these tools monitor sleep EEG (often just a single signal [Ko et al., 2015]), there is a need to create systems for single channel EEG based automatic sleep staging. We explore the option of achieving automatic sleep staging on the basis of a single EEG or EOG channel. Using a single signal permits to simplify the research setup and increases subject comfort.

Polysomnogram (PSG) recordings of subjects are the physiological signals that are collected during an entire night of sleep. The PSG is a multivariate system consisting of signal recordings such as electroencephalogram (EEG), electrocardiogram (ECG), electrooculogram (EOG), and electromyogram (EMG), etc. After the recordings, sleep stage scoring is performed on PSG records, usually performed manually by sleep experts. These experts visually evaluate the PSG signals for a specific time frame, and then determine the scores according to various criteria.

The main criteria for this process are based on the guidelines that were first proposed by Rechtschaffen and Kales (R&K) [Rechtschaffen & Kales, 1968], and later developed by the American Academy of Sleep Medicine (AASM) [Iber et al., n.d.]. According to the rules of R&K, a sleep stage can be classified as wake (W), four non-rapid eye movement (NREM) stages (S1–S4), and rapid eye movement (REM). The AASM has grouped the S3 and S4 stages into a single class representing slow-wave sleep (SWS).

In this study, a flexible deep learning model is proposed using raw electroencephalogram (EEG) and electrooculogram (EOG) signals. A one-dimensional convolutional neural network (1D-CNN) is used to extract features from the data. These features are then passed through a multi-layered convolutional network in order to classify them under the labels W, N1, N2, N3, N4, and REM. The performance of the system is evaluated using two public databases (sleep-edf and sleep-edfx). Our model resulted in the highest accuracies of 96.88%, 90.65%, 82.67%, 85.13%, and 78.46% for two to six classes, respectively, using the sleep-edf database. As these scores are reasonably close to those reported from the paper which this study is based (98.06%, 94.64%, 92.36%, 91.22%, and 91.00% [Yildirim et al., 2019]), our recreation of their deep learning architecture was a success.

2 Related Works

There has been a recent growing interest and scientific investment into understanding sleep [Dijk, 2016]. As such, various different ways of approaching the challenge of automating sleep staging have been proposed. Generally they can be divided into two groups based on their technique for feature extraction. Most of the older models were based on hand-engineered feature based methods that require a prior knowledge of EEG analysis to extract the most relevant features, followed by applying conventional machine learning algorithms such as support vector machines, random forests, and neural networks to train the model for sleep stage classification [Mousavi et al., 2019].

Due to the large amounts of data available and implementation of automated feature extraction based methods (such as convolutional neural networks), more recent studies primarily use deep learning algorithms. These types of models have become more popular as they don't require pre-existing knowledge of the subject and are easily scalable.

After reviewing several published studies, we chose to model our architecture following the one described by Yildirim et al. [2019]. This study was especially useful in understanding deep learning networks as it was robust and clearly explained their process. It used several different types of layers, such as a convolutional, pooling, softmax, and dropout layers, which allowed us to familiar ourselves with concepts of a deep layer network in practice. Furthermore, the authors created several different models from the same data, clearly displaying the influences that different properties had. For example, this study evaluated the different accuracies when using the EEG channel compared to the EOG channel. For both of these sections, they also split the sleep labels into 5 different classes. Finally, this study produced results with accuracies of around 90% compared to manually scored hypnograms.

There were several other studies that were extensively looked into, including those conducted by Fernández-Varela et al. [n.d.], Bresch et al. [2018], and Mousavi et al. [2019]. They were not chosen due to a lack of explanation, diagrams, or robustness. These papers provided us with valuable insights which helped us design solutions and include our own modifications to the study which we recreated.

3 Data

To evaluate our proposed deep CNN model, we used the two most common public sleep datasets. The first of one is the sleep-edf [Kemp et al., 2000] dataset, which contains eight healthy males and females' PSG records. These PSG recordings include two EEG channels: one horizontal EOG signal (Fpz–Cz) and one submental chin EMG signal (Pz–Oz). These signals were obtained with a sampling rate of 100 Hz, and each 30-s fragment epoch, as used in conventional clinical practice [Iber et al., n.d.], was scored based on the RK manual. In the dataset, there are also hypnogram files, which contain annotations of sleep patterns for each subject according to PSGs and were used to construct our class labels. These patterns are labeled for sleep stages as W, S1, S2, S3, S4, REM, M, and '?' (not scored). The second dataset, sleep-edfx [PhysioBank, 2000], is the extended version of the sleep-edf database. The sleep-edfx dataset contains the PSG records of 61 subjects. In this work, *.edf records are divided into two different groups: Sc* and St*. The Sc* records include PSG records of a 24-hour routine of the subjects, and the St* records include one-night data from the hospital collected with a miniature telemetry system.

In the pre-processing stage, we cleaned the datasets to get rid of ambiguous entries, namely those labeled 'M' or '?'. As these data segments did not fall into any of our recognized sleep stages or provide us with useful data, we found it safe to exclude them from our study completely. In order to conserve memory and avoid rounding errors, we transferred the data in its native European Data Format (.edf) directly into the numpy setting that we are using to run the program. The PSG recorded its samples in 10e-2 second increments. As we used 30 second epochs, our input matrix had shape 3000 x 1.

When preprocessing the data we first decided to standardize the data on a per patient basis. We had the intuition, that standardizing the dataset on a per-patient basis could lead to an overall higher accuracy level, as signals could have small errors and shifted and scaled through different measurement equipments and environments as well as physical differences between patients.We shifted and scaled the data by subtracting the mean and dividing by the standard deviation of each signal and patient to have the data Gaussian distributed with a mean of zero and a standard deviation of one. We did this to allow for better training with the neural net while still accounting for outliers. To compare if our assumption that standardization on a per-patient basis works better, we also tried a setting where we concatenated first all patient data and standardized them afterwards (more on this in the experiments section).

4 Methods

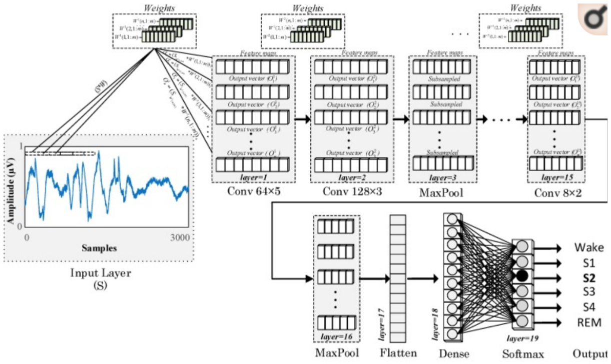
4.1 Architecture

Convolutional Neural Networks (CNNs) are typically used for classifying 2 dimensional images, but can be used for recognition tasks in other settings as well. As the biomedical signals that we are working with are in 1D, we implemented a 1D-CNN. This works the same as a standard CNN, and only differs as its convolutional operation acts in 1D.

A CNN works by taking advantage of known properties of the data in order to apply filters of specific sizes and strides to create feature maps. These feature maps highlight properties of the data that signify that it is part of one group or another. With a final classification layer, the CNN will use information from the maps to give a value corresponding to one labels provided.

We followed the architecture provided by the paper in order to create a 19-layered CNN network. The input data was fed into 2 convolutional layers and then a pooling layer. This series of convolutional and pooling layers was applied several times to further simplify the data. By changing the size of the stride and filters, we were able to reduce the size of the data without losing any information. A dropout layer was applied to prevent overfitting. Finally, a softmax layer was applied to derive the label to which this data point most resembles. A visual depiction, Figure 1, and more details about the architecture can be found in the paper by Yildirim et al. [2019].

Figure 1: 1D-CNN Sleepp Classification Architecture Overview



Our approach for recreating this neural network was to prioritize functionality over optimization. Once we had a functioning network and better understood how it worked, we began playing around with various pieces and observed their influence. We found that most of our changes did not improve our accuracy, so we continued to use the architecture that they presented.

4.2 Tools

Keras was used to implement the model because it was simple to create a working model quickly and because the paper also used it. Google Cloud was used because it allows for GPUs to speed up training. We used the pyedflib library to read EDF files and Tensorboard to allow for easy visualization of training and validation loss and accuracy over each epoch.

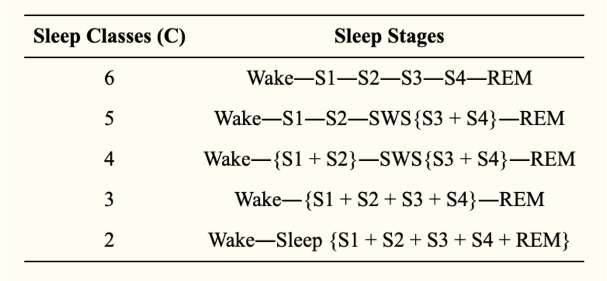
4.3 Choosing a Dataset

Models using the edfx dataset took 14 hours to train over 100 epochs on a Google Cloud GPU. Due to the lack of credits on our machine, we decided to use the sleep-edf dataset so that we could try out different parameters. Due to these resource and time constraints, we reduced the number of epochs we trained for. With more training, we predict that we would have produced higher testing numbers.

4.4 Label Sets

In the paper, the authors split the labels into six different sets, numbered 2-6, that represent the number of labels. Label Set #2 is just sleep or awake stages while Label Set #6 includes wake, S1-4, and REM sleep stages. Figure 2 provides a more detailed breakdown.

Figure 2: Different Label Sets Based on Sleep Stages



Sleep Classes (C)	Sleep Stages
6	Wake—S1—S2—S3—S4—REM
5	Wake—S1—S2—SWS{S3 + S4}—REM
4	Wake—{S1 + S2}—SWS{S3 + S4}—REM
3	Wake—{S1 + S2 + S3 + S4}—REM
2	Wake—Sleep {S1 + S2 + S3 + S4 + REM}

4.5 Workflow

When creating our model, we wanted to be able to control the hyper-parameters of the model easily and efficiently; thus, we created a Python script that allowed us to enter the hyper-parameters, as we called it. Keras allows one to easily create a model and train it within a few functions, so implementing that part was easy. We obscured most of this code in a model class. The harder part was pre-processing the data for the neural net to utilize. We first thought that converting the .edf files to a .csv format, but that proved to be too large. Instead, we read in the .edf files, standardized them, and saved the resulting numpy arrays as .npy binaries. This allowed the model script to load in the binary and feed it into the neural net. Once we tested this locally and set up our VM environment, we transferred all the code and data files to Google Cloud and ran bash scripts to automate training of our models. We saved all of our models as .h5 files and created training and validation loss and accuracy plots for each model we tested (visualized using Tensorboard). To have consistency from run to run, we utilized 42 as our random seed.

4.6 Given Hyper-Parameters

In the paper, it was stated that a learning rate of .0001 was used and the model was optimized using Adam with a decay of .003. We stuck to these parameters as we wanted to experiment with other parameters.

5 Experiments

To get a feel for how our network performed relative to the paper's results, we ran fifteen tests using the edf dataset. We used the FPZ, EOG and EOG+FPZ signals for each label set, hence the fifteen different models. We trained for 50 epochs, 128 as our batch size, and standardized our data on a per patient basis. The results are shown in Table 1.

Table 1: Test Accuracy on EDF Dataset with All Configurations

EDF DATABASE		Test Accuracy	
Classes	Input Type	Paper	Our Model
6	FPZ	89.51	73.89
	EOG	88.28	75.69
	BOTH	91.00	78.46
5	FPZ	90.83	85.13
	EOG	89.78	75.65
	BOTH	91.22	82.40
4	FPZ	91.39	82.67
	EOG	91.88	80.17
	BOTH	92.36	82.01
3	FPZ	94.20	90.65
	EOG	93.76	87.48
	BOTH	94.64	89.60
2	FPZ	98.33	94.16
	EOG	98.02	96.88
	BOTH	98.06	96.44

When testing out different models, we wanted to iterate upon a base model. Since the paper stated that the FPZ signal was the most reliable, we chose our base model to be the model trained on the edf dataset with Label Set #6. The hyper-parameters for this model were the same as previously mentioned (128 batch size, 50 epochs of training, and standardizing data per patient). Henceforth, we will refer to this model as our base model.

5.1 Experiment 1

The first experiment we conducted was testing how the number of epochs affects test accuracy. Due to the lack of computational resources, we ran all of the edf tests for only 50 epochs, rather than 100 as the paper stated. To get a rough estimate of on how much our test accuracy would increase with more epochs, we ran the base model for 100 epochs, keeping the everything else constant. We found that the test accuracy improved by 3.21%, from 73.89% to 77.1%.

5.2 Experiment 2

From class, we learned that dropout layers help networks generalize better. By looking at some of the loss plots in Tensorboard, we were noticing that there was a sizable difference between the loss of the validation and training set. To fix this overfitting problem, we decided to add a dropout layer right before the last layer of the network. This layer would randomly silence 20% of nodes. Running the base model with the added dropout layer created much better results, yielding an accuracy of 78.24%, 4.35% better than the base model. We believed that this was not a fluke, so we continued to use the dropout layer in the rest of our experiments.

5.3 Experiment 3

To further increase our results, we wanted to test batch normalization on our model. This worked very well on assignment 3 of the course and we wanted to see if it has also an effect on our model in terms of faster training / higher accuracy levels after the same number of epochs. We applies batch normalization after the dense layer (layer 19) as well as after all convolutional layers. Keras has a batch normalization function already built on, so implementation was very straight-forward. We tested our new model against the base model including the new dropout layer and it turned out, that batch normalization led to faster growth of accuracy levels, but stayed with an overall accuracy of 75.03%, 3.21% behind the base model. On the other hand, it was quite impressive to see the big potential of batch normalization, when we tried it on a wrongly standardized edfx dataset. The data was standardized to a mean of zero but had a standard deviation of less than e-04. Nevertheless, batch normalization was able to achieve accuracy levels of >80%, where the same model was stuck at an accuracy level of 63%.

5.4 Experiment 4

Due to the success of Experiment 3, we wanted to try this model out on all three signals using the edfx dataset on Label Set #6. Due to the edfx dataset taking a long time to train, we only ran these models for 30 epochs. The results are shown in Table 2.

Table 2: Experiment 3 Trained with the edfx Dataset

	Test Accuracy	
Signal	Our Model	Paper
FPZ	86.61%	89.43%
EOG	82.14%	87.08%
BOTH	83.34%	91.00%

5.5 Experiment 5

Another experiment we wanted to try out was standardizing the data per dataset. In the rest of our models we had standardized our dataset after reading in a file (one file = one patient), but for this experiment we standardized our data as a whole. For more information, check out our explanation of this in the Data section above. When using this technique on the base model without batch normalization, we got an accuracy of 2.15% less than if we had standardized per patient.

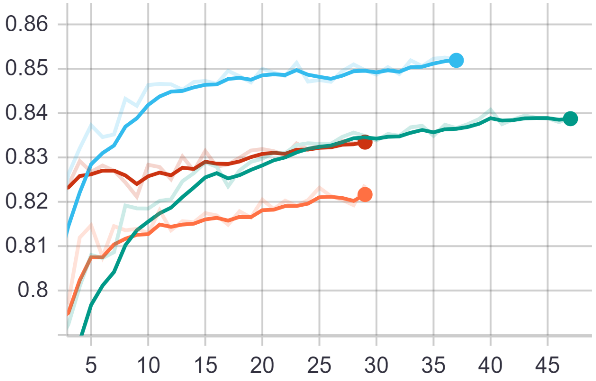
5.6 Experiment 6

We also tried to modify batch size and compared this against our base model. It turned out, that that it takes the models with larger batch size longer to get to a level of accuracy (>70%), but all the levels reached similar levels of loss and accuracy after 30 epochs. We compared our default value of 128 samples from the base model against the same hyper parameter settings and batch sizes of 64, 256, 512 and 1024. For the 1024 model it took 18 epochs to reach a level above 70% accuracy, while it took the model with a batch size of 64 only 2 epochs to achieve the same. Test set accuracy was highest for model with batch size 64 (81.40%), while it was lowest for the model with batch size 512 (77.31%). It would be interesting to investigate the effect of a higher learning rate for the models with larger batch size.

5.7 Experiment 7

When we were looking at the proposed model in the paper, we were wondering why each model was only optimized to accept one signal. We were interested in seeing what would happen if we increased the number of input features in the model by feeding in the FPZ and EOG signals at the same time. Thus, we specified an option to build a model with a two-dimensional feature input, making our input data by 3000 by 2 rather than 3000 by 1. When testing, we used the same hyper-parameters as Experiment 3. On the edf dataset, we got an accuracy of 81.75%, which was 6.72% better than Experiment 3 using the FPZ signal. We also tried running the model on the edfx dataset, but due to lack of computational resources, we had to cut training off after 38 epochs. This model has an accuracy of 85.22% on the validation set, which was higher than accuracy on the EOG and both signal, but less than the FPZ signal (see Figure 3).

Figure 3: Compare development of validation set accuracy: Experiment 7: blue, same settings but 1D with the FPZ / EOG / FPZ + EOG signal: green / orange / red



5.8 Summary of Experiments

Although we weren't able to match the accuracy of the paper in the first set of our testing, we were able to conduct experiments that increased the accuracy of our model. We found that another dropout layer and batch normalization raised accuracy with the FPZ signal (by 4.35% and 3.21% respectively). Using this intuition, we tested our upgraded model on the edfx dataset and found results much more in line with the paper, even with training for only 30 epochs instead of the recommended 100. We also tried standardizing the data relative to the whole dataset but that didn't perform as well as previous models. When testing batch size we noticed that smaller batch sizes work better and converge faster. Inputting two features into the neural net also didn't improve accuracy. Our model was relatively robust as it achieved pretty similar results without being fully tested for as many epochs in the paper.

6 Conclusions

We have successfully created a 1D CNN to classify sleep stages with an accuracy of 78.46% for 6 class classifications and 96.88% for 2 class classifications using single channel EEG, EOG, or both signals. With our final models, we focused on comparing results from the smaller edf database, as we could run extensive tests on it. While we still did use the sleep-edfx database, its large size made it difficult to run with our available computational power. With more processing power and time to train our networks with more data, we predict that we would have higher accuracies.

In conducting this study, we have learned about working with neural networks and the research process as a whole. We spent a lot of time figuring how to best preprocess the data, removing chunks of data that were mislabelled or incorrectly formatted. As we wanted to run our program with several different settings, another challenge was to make the model malleable and flexible. While easy in theory, this proved to cause a lot of annoying little issues. In terms of future experimentation, modifying the learning rate and optimizers would be interesting to investigate. Investingating more initialization strategies would also improve the network as we noticed a big shift in accuracy when changing our initialization strategy. In summary, we have learned that nearly everything ends up taking longer than expecting and things often don't work due to the reasons that we originally expected. Research is a tedious process, so don't sleep on it.

References

Bresch, E., Grossekathofer, U., & Garcia-Molina, G. (2018). Recurrent deep neural networks for real-time sleep stage classification from single channel eeg. *Frontiers in computational neuroscience*, *12*, 85.

Dijk, D. (2016, Dec). *Sleep research in the near future: a passing outlook.* John Wiley Sons, Ltd (10.1111). Retrieved from `https://onlinelibrary.wiley.com/doi/full/10.1111/jsr.12489`

Fernández-Varela, I., Athanasakis, D., Parsons, S., Hernández-Pereira, E., & Moret-Bonillo, V. (n.d.). Sleep staging with deep learning: a convolutional model. In *Proceedings of the european symposium on artificial neural networks, computational intelligence and machine learning (esann 2018).*

Iber, C., Ancoli-Israel, S., Chesson, A., & Quan, S. (n.d.). The aasm manual for the scoring of sleep and associated events: Rules, terminology and technical specification, (2007). *Westchester American Academy of Sleep Medicine*, 17–9.

Kemp, B., Zwinderman, A. H., Tuk, B., Kamphuisen, H. A., & Oberye, J. J. (2000). Analysis of a sleep-dependent neuronal feedback loop: the slow-wave microcontinuity of the eeg. *IEEE Transactions on Biomedical Engineering*, *47*(9), 1185–1194.

Ko, P.-R. T., Kientz, J. A., Choe, E. K., Kay, M., Landis, C. A., & Watson, N. F. (2015). Consumer sleep technologies: a review of the landscape. *Journal of clinical sleep medicine*, *11*(12), 1455–1461.

Marshall, L., Helgadóttir, H., Mölle, M., & Born, J. (2006). Boosting slow oscillations during sleep potentiates memory. *Nature*, *444*(7119), 610.

Marshall, L., Mölle, M., Hallschmid, M., & Born, J. (2004). Transcranial direct current stimulation during sleep improves declarative memory. *Journal of Neuroscience*, *24*(44), 9985–9992.

Mousavi, S., Afghah, F., & Acharya, U. R. (2019). Sleepeegnet: Automated sleep stage scoring with sequence to sequence deep learning approach. *arXiv preprint arXiv:1903.02108*.

Panossian, L. A., & Avidan, A. Y. (2009). Review of sleep disorders. *Medical Clinics of North America*, *93*(2), 407–425.

PhysioBank, P. (2000). Physionet: components of a new research resource for complex physiologic signals. *Circulation. v101 i23. e215-e220*.

Rechtschaffen, A., & Kales, A. (1968). A manual of standardized terminology, techniques and scoring systems for sleep stages of human subjects (usphs no 204) us government printing office. *Washington, DC*.

Vogel, G. W., Vogel, F., McAbee, R. S., & Thurmond, A. J. (1980). Improvement of depression by rem sleep deprivation: New findings and a theory. *Archives of General Psychiatry*, *37*(3), 247–253.

Yildirim, O., Baloglu, U. B., & Acharya, U. R. (2019). A deep learning model for automated sleep stages classification using psg signals. *International journal of environmental research and public health*, *16*(4), 599.

A Knowledge Acquired

A.1 Martin Kjäer

1. **Understand how to build a "real world" CNN (one with a practical use case).** After learning the basics of deep learning during the lectures and assignments, it was interesting to also try experiment in another area and a practical use-case. During the course of the project I learned to use a 1D convolutional network and how a typical architecture for a problem like this could look like.

2. **Get an idea of training time / need to scale on a cloud platform.** After we made the basic model work, we quickly found out, that it will not be practical, to run this model on our local machines. I learned, how to work with the google cloud platform and got a better feeling how much time it takes to train our model on a GPU. Furthermore I got a better understanding on how much it costs to train models on a cloud platform.

3. **Learn to run different experiments and improve a given deep neural net.** I learned a lot when we experimented with the neural net after it was basically working to train it also in the cloud and we were trying many things to improve accuracy. I leared to

A.2 Lukas Kulbis

1. **Learn how to efficiently approach (deep learning) research papers.** Due to reading several academic studies on deep learning networks for automated sleep staging, I am now more familiar with the research process in general. I understand the process of introducing the problem, describing the data, explaining the methods, reviewing the results and concluding. While searching for a model to follow, I focused on comparing the *methods* section to find one which would be suitable for us, as this was the main difference.

2. **Understand how a neural net can classify atypical data.** By using single channel signals from a PSG recording, I was forced to learn how a deep learning network works on something other than standard 2D RGB images. After analyzing the diagrams and looking at the raw data, I have come to understand how the neural net is able to generate classifications from a simple recording.

3. **Become familiar with scalable deep learning tools.** Throughout this project I have been exposed to several tools which are used by professionals on large scale studies. After reading documentation and through trial and error, I am now able to navigate through the Google Cloud Platform and use the Tensorboard software.

A.3 Rahul Shah

1. **Learn Keras** I was in charge of creating the model and making sure it ran correctly. I read the Keras documentation and implemented the model to be able to be configured with many different hyper-parameters. I also worked on refomatting the training data to be input in to the net.

2. **Observe the positives/negatives of building a deep network (nineteen layers).** I was unfamiliar with deep networks before this project and this gave me an up-close view of the pitfalls and positives of deep networks. I realized that they are able to be highly customized and really through, as this particular network was able to reduce a 3000 input into a 64 hidden layer representation. However, I noticed that due to the depth of the network, it depended heavily on initialization even when using batch normalization.

3. **Decide which hyper-parameters to use given input data and number of output labels.** I think that finding the right hyper-parameters is more of a trial-and-error sort of ordeal and this project really was like that. We experimented with a variety of different strategies, and in general, the ones that reduced generalization performed the best, such as batch normalization and dropout

layers. Smaller batch sizes and more update steps are significantly helped the network, and I think this was because our learning rate was very low.