Pre-Interview Tasks

Luke T. Patterson

ltpatterson2013@gmail.com

1. Data Exploration and Preprocessing

The data for the 2017 PhysioNet/Computing in Cardiology Challenge was provided as a zip file, training2017.zip, containing a series of .mat and .hea files as well as two .csv files named REFERENCE.csv and REFERENCE-original.csv. The .mat files contain the time series data for each of the recordings, and the .hea files are header files containing information about the recordings, including sampling rate, when the recording was taken, and the number of samples. The two reference files contained the mapping for each ECG recording to its class. I believe the REFERENCE-original.csv was perhaps the file used during the competition, and the REFERENCE.csv is an updated mapping of classes, but I didn't see any information specifying this.

To explore the data using Python, I constructed a pandas data frame consisting of the file name, ECG data, and class label. I then picked this data frame to load and manipulate the complete dataset more easily later. To get a better idea of the class distribution and the sequence length, I created two plots.

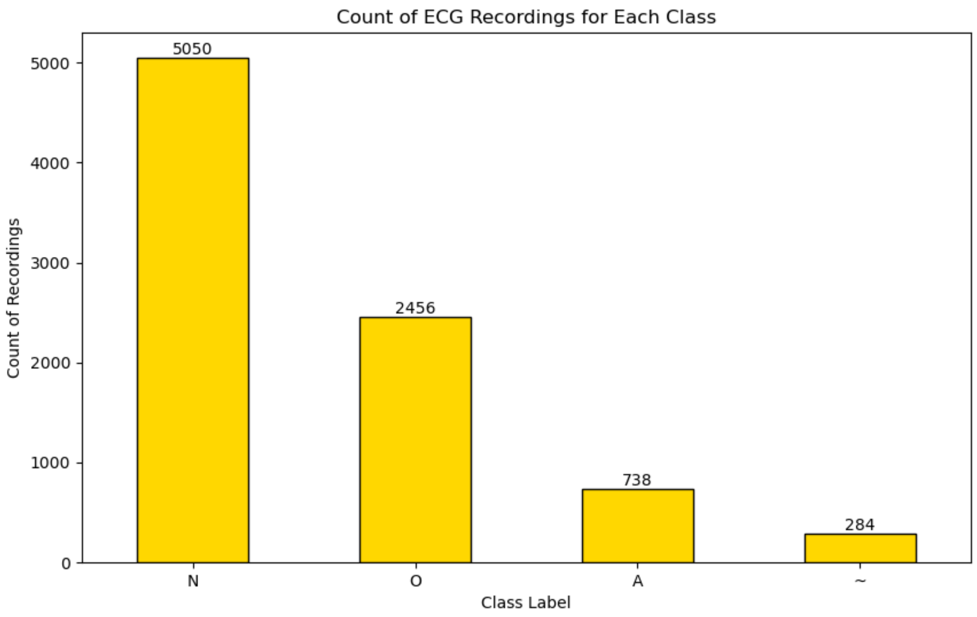


Figure 1: Class Distribution

Dec 2023

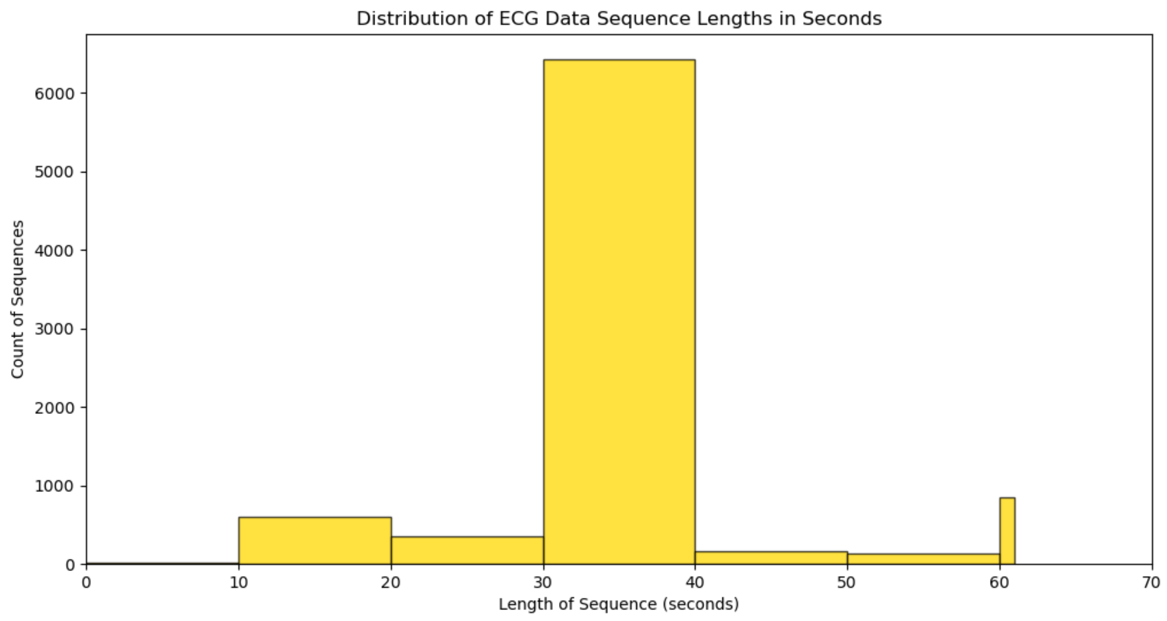


Figure 2: Sequence Length Distribution

From these two figures, I was able to discern that there was a heavy class imbalance in the data and that the sequences were typically between 30 and 40 seconds in length, but some were up to just over 60 seconds in length.

In reviewing some literature regarding ECG classification, I noticed some individuals successfully used spectrograms and scalograms as input for their classifiers. Having worked with spectrograms in the past for a music informatics class, I decided to use them as input for my classifier. I began processing the data and splitting up a training, testing, and validation set. To process the data, I decided to pad using 0s or trim the recordings to 60 seconds to include the entire dataset despite the length. Then, I converted them into a spectrogram before performing the train, test, and validation split. To construct the training and testing set, I used a stratified 80/20 split and then split the resulting training set into training and validation with a 75/25 split (so the validation set would consist of 20% of the entire dataset).

The exploration and preprocessing of the data can be found in the file main.py.

2. Pre-Trained Transfer Learning Model

I have had limited experience with transfer learning, so I felt this was an excellent opportunity to learn more about it by adapting a model. In reviewing the literature, I found a paper titled "A Transfer-Learning Based Ensemble Architecture for ECG Signal Classification," which leveraged the pre-trained model InceptionResNetV2 and was very successful in classifying their data, so I decided to use this as my base model.

For my model adaptation, I removed the top layer of InceptionResNetV2, which was explicitly designed for the ImageNet challenge. I added a Global Average Pooling layer as well as a dense layer and softmax output layer.

2.1. Hyperparameter Tuning

For the model's design, I tuned the following parameters using an automated Bayesian sweep tracked through the service WandB.

- Number of neurons in the dense layer.
- Activation function of dense layer.
- Optimizer type.
- Optimizer learning rate.
- Batch size.

Due to time constraints, I only allocated ten epochs per sweep run. Here is a visual representation of the top 15 runs.

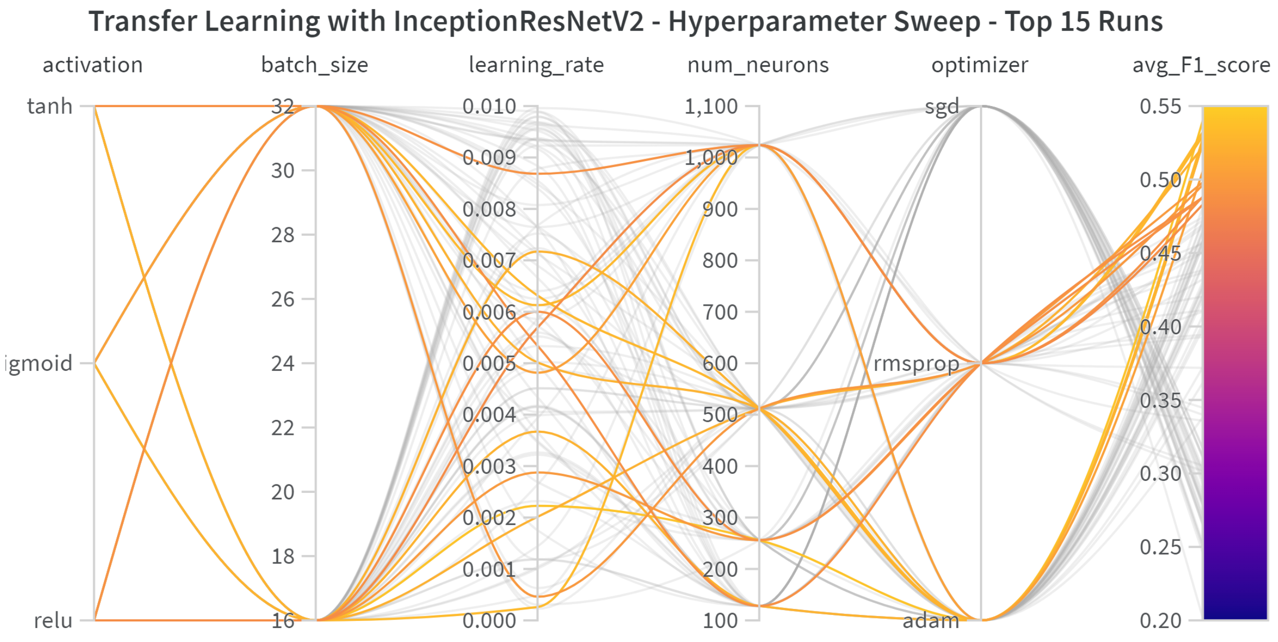


Figure 3: Top 15 Sweep Runs

For the sweep and training, I designed a custom Keras callback that would allow me to grade the model based on the competition's F1 score to optimize how the model was performing. This callback function can be found in f1_sweeping_callback.py while the code for the sweep itself can be found in sweeping.py. Reviewing the information from the sweep and WandB's provided correlation measurements, the activation functions seemed to perform similarly, but sigmoid had a slight advantage. The learning rate performed best between 0.002 and 0.005. The fully connected layer performed best with around 500 or fewer neurons. Both rmsprop and adam optimizers performed well, but adam had a higher positive correlation according to the statistics compiled by WandB. The batch size didn't seem to ultimately impact the training too much, so for my GPU's sake, I decided on a smaller one.

For the final model, I would train using the following parameters.

- 500 neurons in the dense layer.
- Sigmoid activation function.
- Adam optimizer.
- Batch size of 16.

3. Model Evaluation

I trained the final model for 100 epochs, and with my custom callback for each epoch, I would use the competition's metrics to calculate the F1 score, and if there was an improvement, I would save the model. The callback also has the ability to log a confusion matrix image to WandB's website for monitoring online during training when a model is saved, which can allow me to check how the model is performing across classes as it trains. This callback function can be found in the file f1_training_callback.py.

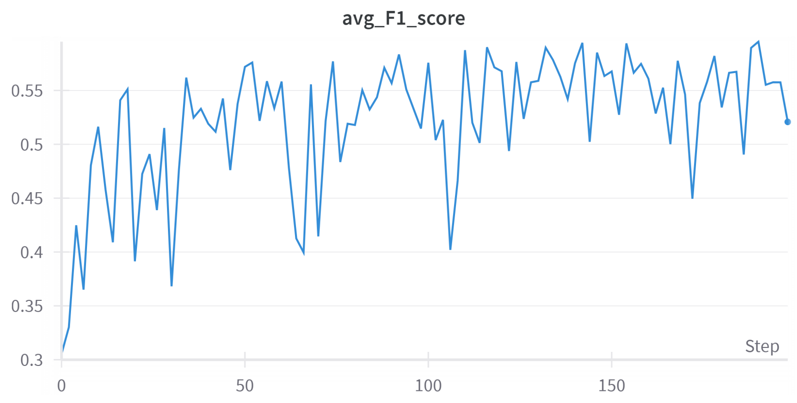
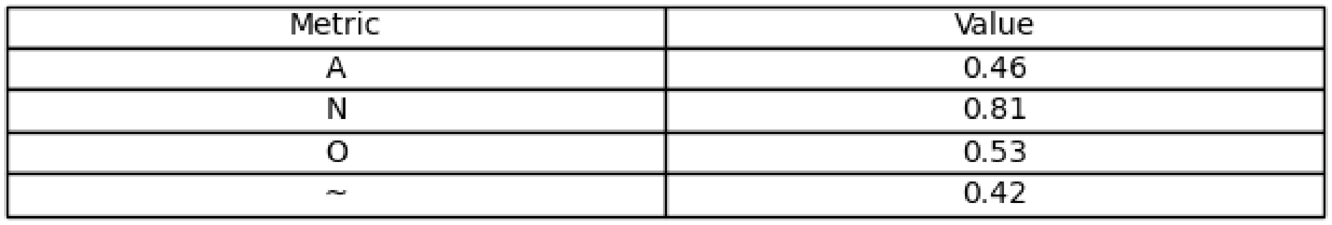


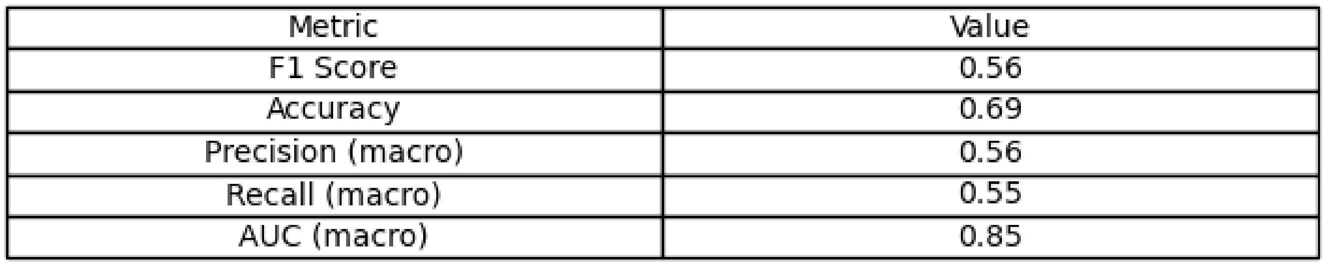
Figure 4: Average F1 Score for each Training Step

After training the model had achieved an F1 score of 0.5953 on the validation data. I constructed an evaluation script to test the model on the holdout test dataset, then constructed tables and visualizations with the metrics.



Metric	Value
A	0.46
N	0.81
O	0.53
~	0.42

Figure 5: F1 Score for each Class



Metric	Value
F1 Score	0.56
Accuracy	0.69
Precision (macro)	0.56
Recall (macro)	0.55
AUC (macro)	0.85

Figure 6: Overall Metrics

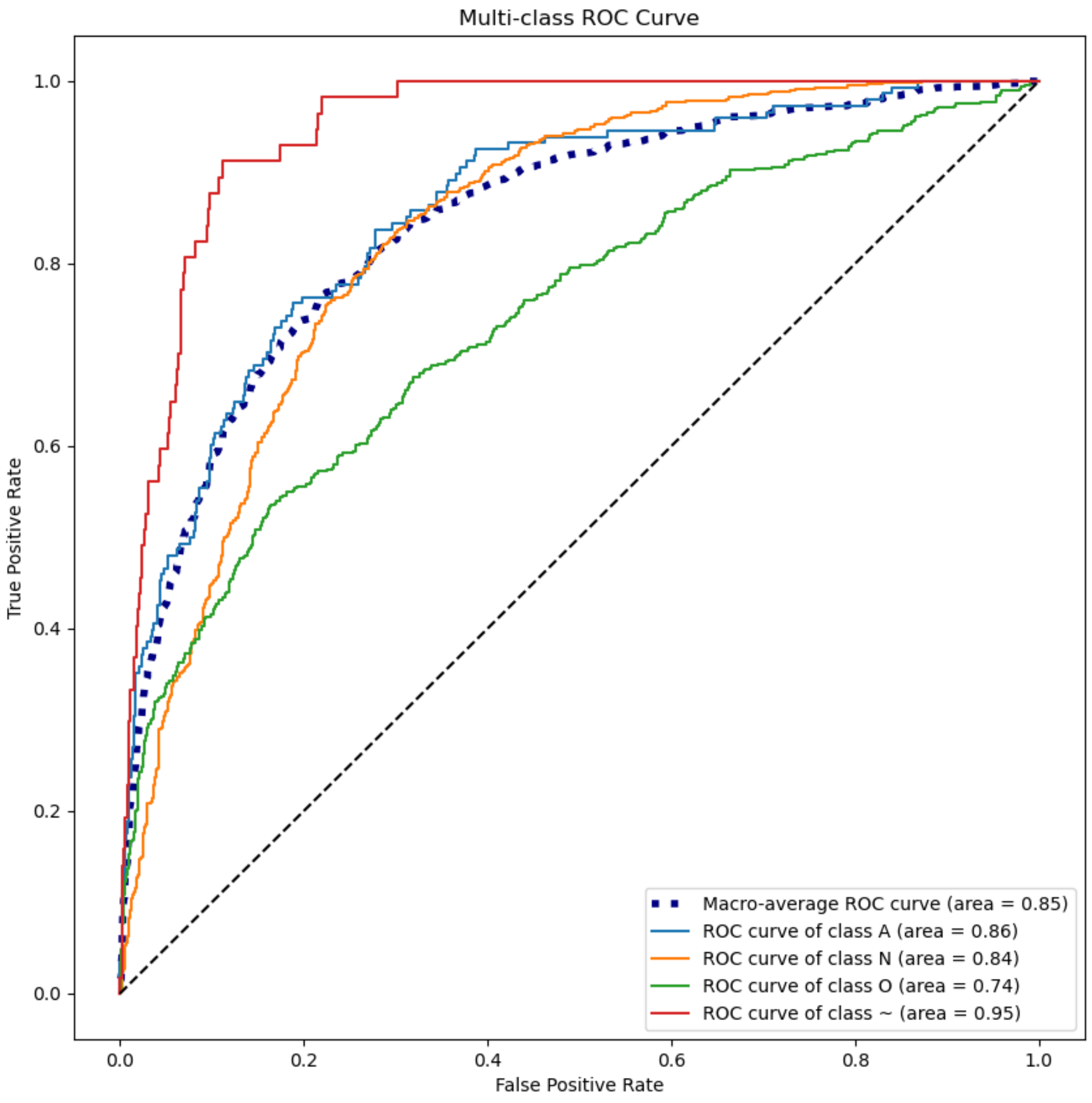
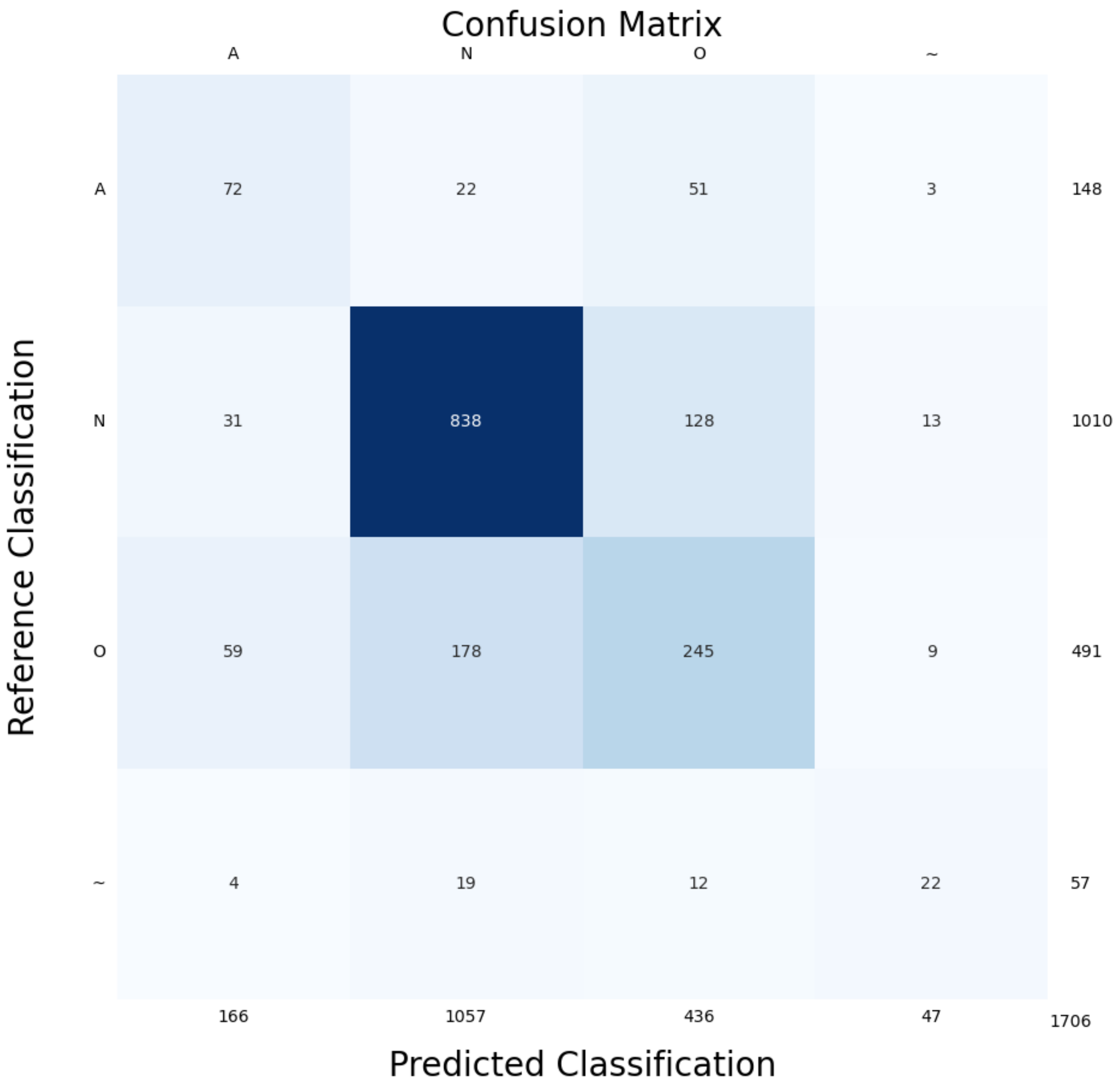


Figure 7: Macro ROC



	A	N	O	~	
A	72	22	51	3	148
N	31	838	128	13	1010
O	59	178	245	9	491
~	4	19	12	22	57
	166	1057	436	47	1706

Figure 8: Test Data Confusion Matrix

The overall model performance was not exceptional, achieving only a 0.56 F1, but this was comparable to some of the 2017 challenge entries. The ROC curve was better than I had anticipated but I feel the F1 scores are more indicative of the actual performance of the model. The individual class F1 scores appear to be ordered in the prevalence of each class in the dataset leading me to believe that the class imbalance impacted the performance of the model. The test data confusion matrix also leads me to a similar conclusion. All of the evaluation code can be found in the file evaluation.py.

4. Grad-CAM Model Interpretation

I have never previously used the Grad-CAM algorithm for model interpretation in the past so this was a learning experience for me. It was interesting to learn about and I feel it will be very helpful for evaluating models in the future. Luckily I found a helpful blog post and was able to present a straightforward implementation that I was able to get working with the spectrogram images I had constructed from the ECG data.

(https://deeplearningofpython.blogspot.com/2023/05/Gradcam-working-example-python.html)

The code for my implementation can be found in gc_implementation.py. Unfortunately, the results were discouraging since it seems my model was greatly impacted by the padding of the data before converting them into spectrograms as can be seen in the resulting images below with one from each class.

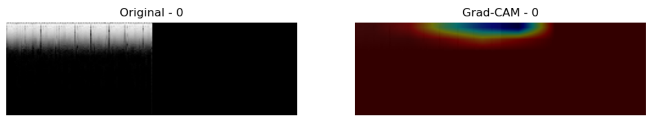


Figure 9: A Class Grad-CAM Overlay

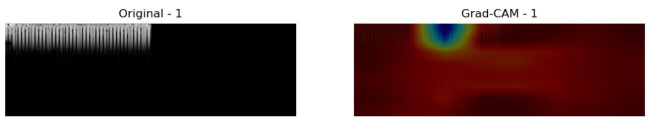


Figure 10: N Class Grad-CAM Overlay

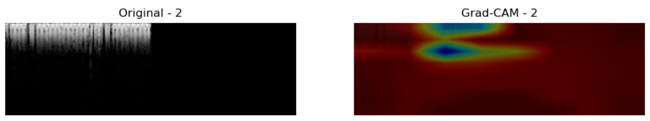


Figure 11: O Class Grad-CAM Overlay

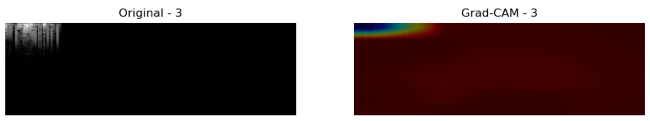


Figure 12: ~Class Grad-CAM Overlay

While this discovery was discouraging, it gives a clear indication that the model could improve with better preprocessing techniques.

5. Future Work

Given more time to refine this project, I am confident that several enhancements could be made to both the model architecture and data preprocessing techniques. Exploring scalograms and the features offered by wavelet transforms could provide a better representation of the ECG data, potentially improving the model's classification ability over spectrograms. A key area for improvement is the data preprocessing approach. The current method of padding the data may inadvertently influence the model's decision-making process. Alternative strategies, such as segmenting the data into windows, augmenting with random noise or time warping, and resampling, could improve the model's ability to recognize patterns in the ECG data. Further, architectural modifications, such as using ensemble models could greatly improve the classification ability.

The code I constructed could also be further refined to make the addition of new data easier and make the processing of the dataset more streamlined using data generators. Currently, there may be issues with training on lower-end hardware because a large chunk of the dataset is loaded into memory, and this caused issues with the custom callbacks, which had to be modified to process the validation data in chunks to avoid memory errors. In future projects, I would also dedicate additional resources to sweeping the hyperparameters of any improved model architecture since 10 epochs was long enough to get an idea of the model performance, but with a higher number of epochs per run and more time dedicated to the sweep I feel the selected parameters could be improved.

6. Conclusion

Thank you again for the opportunity to interview for this position at your lab and for taking the time to review the work I've done here.

Below I will list and describe the Python files that will be included in the project.

- main.py - This program performs data exploration and is also responsible for training the final model configuration.
- sweeping.py - This script is constructed to begin a hyperparameter sweep using the Weights and Biases online machine learning monitoring platform.
- f1_sweeping_callback.py - This file contains a class for a custom Keras callback which is used during the sweep for logging and evaluation purposes.
- f1_training_callback.py - This file contains a class for a custom Keras callback which is used during the final model training.
- gc_implementation.py - This file contains my Grad-CAM implementation.
- evaluation.py - This program calculates several evaluation metrics and constructs visualizations to present them.
- preprocessing.py - This file contains a function that was commonly used across files for preprocessing the model's input data.

Python libraries used in this project include:

- pandas
- scipy
- os
- datetime
- gc
- pickle
- matplotlib
- seaborn
- numpy
- sci-kit learn
- tensorflow
- opencv-python
- wandb

7. References

Clifford GD, Liu C, Moody B, Li-wei HL, Silva I, Li Q, Johnson AE, Mark RG. AF classification from a short single lead ECG recording: The PhysioNet/computing in cardiology challenge 2017. In 2017 Computing in Cardiology (CinC) 2017 Sep 24 (pp. 1-4). IEEE. https://doi.org/10.22489/CinC.2017.065-469

J. Huang, B. Chen, B. Yao and W. He, "ECG Arrhythmia Classification Using STFT-Based Spectrogram and Convolutional Neural Network," in IEEE Access, vol. 7, pp. 92871-92880, 2019, doi: 10.1109/ACCESS.2019.2928017.

T. B. Ovi, S. S. Naba, D. Chanda, and Md. S. H. Onim, “A Transfer-Learning Based Ensemble Architecture for ECG Signal Classification.” arXiv, 2022. doi: 10.48550/ARXIV.2207.00002.

Y. Dakshina, “GRADCAM(gradient-weighted class activation mapping) working, applications and its implementation in Python,” Deep Learning of Python, https://deeplearningofpython.blogspot.com/2023/05/Gradcam-working-example-python.html (accessed Dec. 9, 2023).