ECG Heartbeat Categorization Report

Yuxing Zhang

1 Data Processing

1.1 Exploratory Data Analysis

The MIT-BIH Arrhythmia Dataset has already been normalized and single heartbeats have been extracted [5]. Each example has 187 features. Since training set and test set have been provided, I created a validation set from 10% of training data, retaining the same class distribution. The number of examples belonging to each class are summarized in table 1.

Table 1: Distribution of classes.

	N	S	V	F	Q	Total
Train	65224	2001	5210	577	5788	78800
Validation	7247	222	578	64	643	8754
Test	18118	556	1448	162	1608	21892

Clearly, the data is not balanced, meaning we need to do some data augmentation before a classifier can be trained. Before going to that step, I plotted an example from each class just to have a rough idea what different heartbeats look like, as in figure 1.

1.2 Data Augmentation

Approaches to augmenting time series data can generally be divided into two groups [4], random transformation and oversampling. Due to the heavy imbalance of our data, I focused on oversampling the minority classes and undersampling the majority classes. Specifically, I applied synthetic minority oversampling technique (SMOTE) [1] to create a balanced training dataset. SMOTE creates synthetic examples by interpolating between each example and its nearest neighbours. Compared to naive oversampling, SMOTE generates new data unseen in the original dataset, which could potentially help the classifier better generalize. While there are many metrics for nearest neighbour search, e.g. Euclidean or

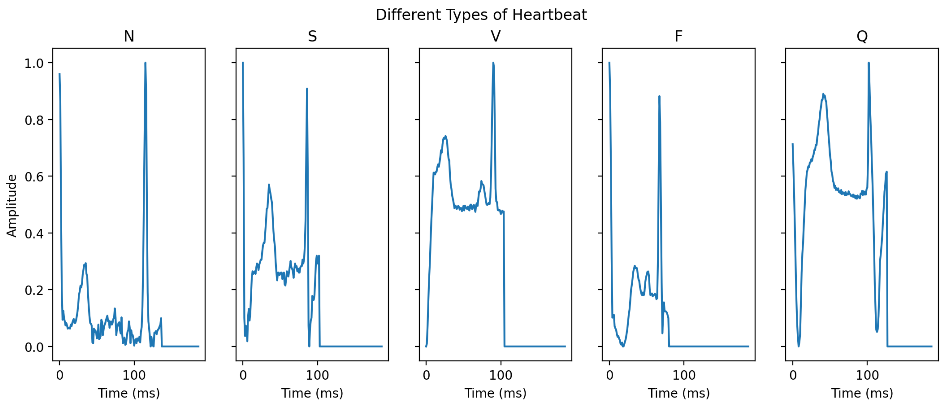

Figure 1: An example heartbeat from each classes.

$L1$ distance, I find dynamic time warping (DTW) [7] particularly well-suited for measuring similarity between temporal sequences due to its shift invariance.

The balanced training dataset is created the following way.

- Class N, V and Q are undersampled to 3000 examples each
- Class S is first oversampled to 4002 examples via SMOTE (ratio = 1) and then downsampled to 3000 examples
- Class F is first oversampled to 3462 examples via SMOTE (ratio = 5) and then downsampled to 3000 examples

I implemented both DTW and SMOTE from scratch. Figure 2 shows a sample of original and synthetic data from class S.

1.3 Feature Engineering

While it is viable to select certain features from the raw training data and train a classifier such as random forest or SVM, it requires expert knowledge to extract the most relevant features. Hence, I took the preferred approach in the deep learning community by training an end-to-end deep model, i.e. representation learning.

It can been seen in figure 1 that the tails are very flat. This, coupled with the fact that the length of each sequence is 187, encouraged me to trim the sequence to a length that facilitates training a convolutional network. Specifically, I truncated each sequence to a length of 160 by removing the tailing 27 features.

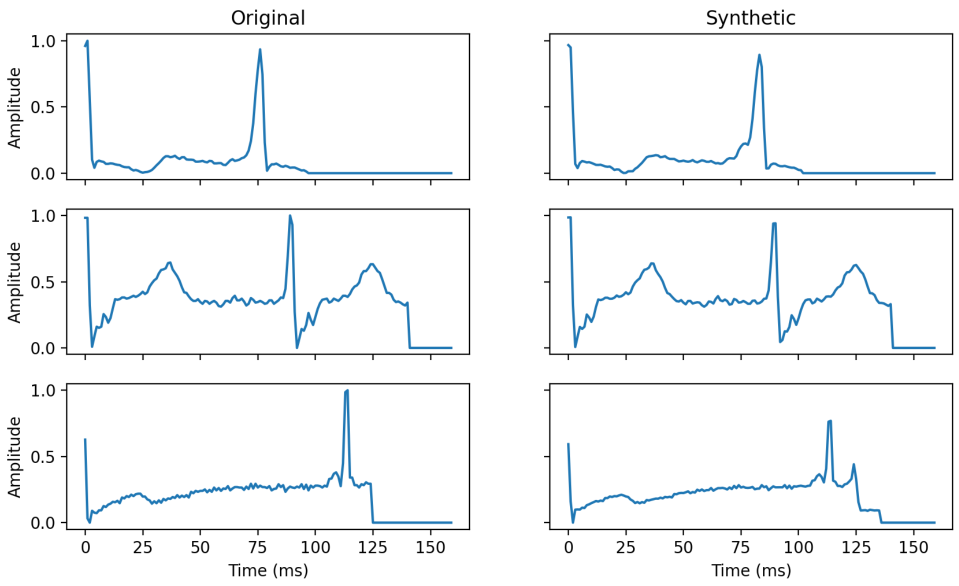

Figure 2: Original and synthetic examples from class S. They look very similar but have subtle differences.

2 Model Training

2.1 Model Architecture

Sequence data can be modelled with various neural networks, including RNN, CNN and attention based networks such as Transformers. CNNs in particular has seen success in arrhythmia detection [6]. I tried to strive for simplicity by using an all convolutional net [8]. The model architecture is summarized in table 2.

Here, the parameters in convolutional layers denote input channel, output channel, filter size, stride and zero padding respectively. Neural networks have very high representational capacity due to their parameter count. Hence regularization methods are necessary for model generalization. I used batchnorm [3] after each convolutional layer to mitigate overfitting. I also considered dropout [9] but did not see apparent benefits. The last layer is equivalent to a fully connected layer. Also, no activation is followed after the last convolutional layer since it is more numerically stable to use logits directly to compute cross entropy loss.

Table 2: Model architecture

Layer	Output shape (channel, length)
	input (1, 160)
Conv(1, 4, 4, 2, 1) BatchNorm ReLU	(4, 80)
Conv(4, 8, 4, 2, 1) BatchNorm ReLU	(8, 40)
Conv(8, 16, 4, 2, 1) BatchNorm ReLU	(16, 20)
Conv(16, 32, 4, 2, 1) BatchNorm ReLU	(32, 10)
Conv(32, 64, 4, 2, 1) BatchNorm ReLU	(64, 5)
Conv(64, 5, 5, 1, 0)	(5, 1)

2.2 Training and Evaluation

I trained my models on AWS. I did a grid search for the learning rate $lr \in \{0.01, 0.05, 0.1, 0.25\}$ and implemented early stopping with the maximum epoch set as 300. Early stopping is a common technique to combat overfitting. The rational behind it is that when the model overfits the training data, the validation loss will start increasing. The learning curves for all models are shown in figure 3. It can been seen that the validation loss dose not stop at its minimum. This is because I implemented a tolerance window to accommodate the natural fluctuation.

To select the best model, I used Area Under the Receiver Operating Characteristic Curve (ROC AUC) to measure each model's performance. Model accuracy is not proper due to class imbalance. As can be seen in table 3, AUC of each model is very close, $lr = 0.01$ being slightly better.

Having selected the model trained with $lr = 0.01$, I evaluated it on the test set, resulting in an AUC of 0.9894211879292485. The confusion matrix is shown in figure 4.

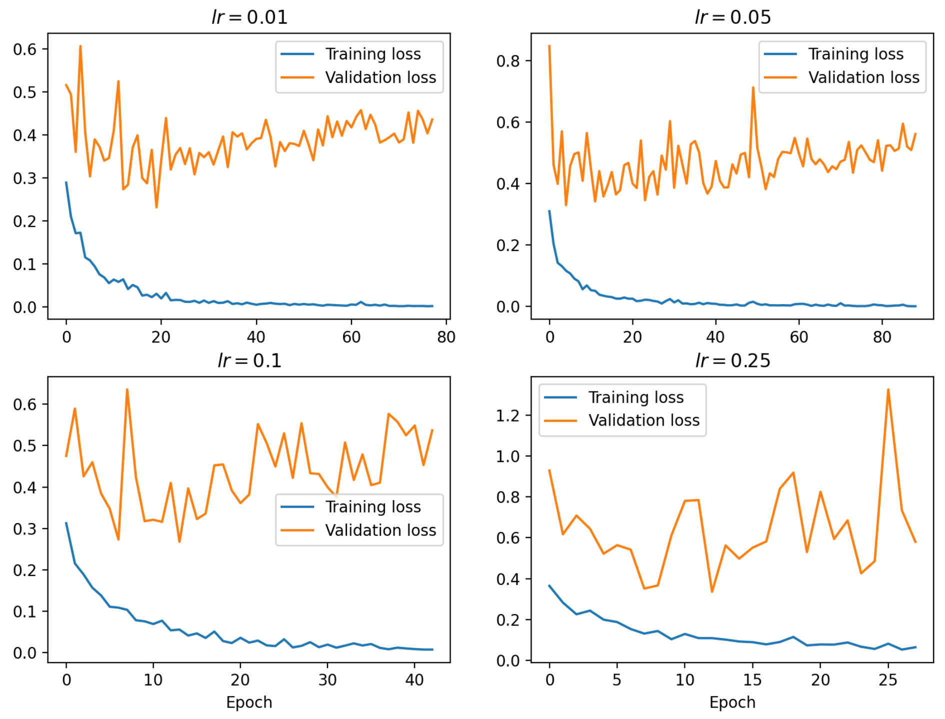

Figure 3: Learning curves during my grid search. Training stops at different epoch depending on the learning rate.

Table 3: AUC of models trained with different learning rates.

lr	AUC
0.01	0.9946872791309567
0.05	0.9919687198761841
0.1	0.9928475520006288
0.25	0.988960362208835

3 Discussion

3.1 Model Accuracy

Accuracy is an interesting topic in classification with unbalanced data, as a blind prediction to the majority class can yield high accuracy. To have a clearer view of the accuracy of my

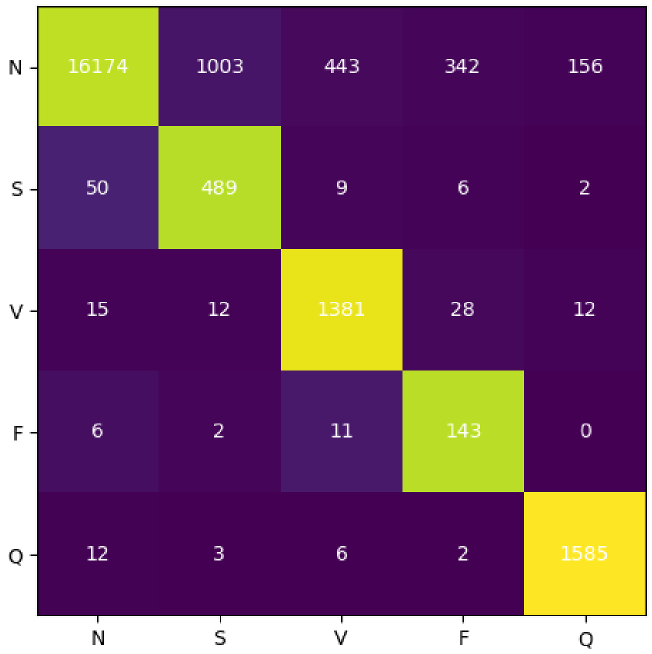

Figure 4: Confusion matrix of model predictions.

model, I calculated both the overall accuracy and per class accuracy, results in the balanced data row in table 4. We can see the model reaches relatively consistent accuracy among the classes. I speculated that if trained on the original imbalanced data, the accuracy would be highly inconsistent, with much high score for the majority classes. I trained such a model with the same learning rate $lr = 0.01$ and the results, shown in the original data row in table 4, justified my thoughts. This also explains why accuracy is not a good performance measure when data is not balanced.

Table 4: Model overall accuracy and per class accuracy.

	N	S	V	F	Q	Overal
Balanced data	0.89	0.88	0.95	0.88	0.99	0.90
Original data	0.99	0.77	0.95	0.78	0.98	0.98

3.2 Limitations and Potential Improvements

First, regarding synthetic data, while SMOTE is effective, it is a very slow process due to dynamic programming needed to calculate every instance of DTW. In fact,SMOTE took the most of time during my experiments, outweighing model training by a large margin. I could use Euclidean distance instead, which could easily be implemented as matrix multiplication, at the cost of losing shift invariance provided by DTW. One approach I am particularly interested in is synthesizing new data with generative models, such as GANs. There has been successful attempt [11] in this direction.

I designed the model architecture following a simplicity principle. It can potentially be improved with a more complex structure, such as by adding skip connection [2]. Recently, attention based models have seen success in every ML field, make them a potential candidate for the problem of ECG data classification.

Last but not the least, MIT-BIH is a very well studied dataset, but quite limited in size. A strong classifier could be created by first pre-training it on a much larger dataset [10].

3.3 Model Deployment

Models can be deployed in various ways, such as on the cloud, on-premises or on edge devices, depending on the use cases. There are many challenges and considerations in model deployment. E.g. data can be fed online or in batches. There could be a drift in the coming data so we need strict monitoring to prevent the model from degrading. We also need to carefully decide on the tools to use as there is often an ecosystem around a certain tool. E.g. Pytorch has a deployment system called TorchServe, so does Tensorflow with TFX. Cloud vendors also have an array of services to meet various needs. E.g. AWS have SageMaker for model development and S3 for data store. As for model serving, it can be done via HTTP by REST apis, or served locally like on edge and embedded devices. Some important areas regarding deployment include:

Scalability Horizontal scaling should be preferred over vertical scaling. We could deploy our model on a public cloud and implement automatic scaling rules, e.g. with Kubernetes. Load balancing and caching mechanisms are also necessary to improve efficiency.

Versioning Versioning in ML is different from software engineering in that besides codes, we also need to version models and data. Github cannot handle files larger than 50MB. However, there are tools specially designed for versioning ML projects, such as DVC.

Monitoring Monitoring is very import in production environment as any change in the data or our models can have potentially disastrous outcome. Luckily, every cloud vendor has solid monitoring service. There are also open source options such as Prometheus.

Retraining Following CI/CD practices, we can set up automatic retraining. A/B tests can be utilized before the new model gets served. If our model is deployed on edge devices, we can instead deploy a training script so that retraining will happen on the edge.

Privacy In many cases, medical data in particular, privacy is a serious concern. We can employ privacy training, such as differential privacy, in model development process. We can also use federated learning, such as by deploying the model on edge devices.

References

[1] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arxiv 2015. *arXiv preprint arXiv:1512.03385*, 14, 2015.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[4] Brian Kenji Iwana and Seiichi Uchida. An empirical survey of data augmentation for time series classification with neural networks. *Plos one*, 16(7):e0254841, 2021.

[5] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. Ecg heartbeat classification: A deep transferable representation. In *2018 IEEE international conference on healthcare informatics (ICHI)*, pages 443–444. IEEE, 2018.

[6] Pranav Rajpurkar, Awni Y Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y Ng. Cardiologist-level arrhythmia detection with convolutional neural networks. *arXiv preprint arXiv:1707.01836*, 2017.

[7] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[8] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[10] Shawn Tan, Guillaume Androz, Ahmad Chamseddine, Pierre Fecteau, Aaron Courville, Yoshua Bengio, and Joseph Paul Cohen. Icentia11k: An unsupervised representation learning dataset for arrhythmia subtype discovery. *arXiv preprint arXiv:1910.09570*, 2019.

[11] Fei Zhu, Fei Ye, Yuchen Fu, Quan Liu, and Bairong Shen. Electrocardiogram generation with a bidirectional lstm-cnn generative adversarial network. *Scientific reports*, 9(1):1–11, 2019.