Machine Learning for Healthcare Project 1 Report

Arda Arslan
aarslan@student.ethz.ch

Gökberk Özsoy
goezsoy@student.ethz.ch

Sebastian Müksch
smueksch@student.ethz.ch

1 INTRODUCTION

A major risk to personal health, heart diseases can be diagnosed by observing different patterns in so-called electrocardiogram (ECG) signals taken of the patient, a form of time-series data. In order to assist physicians in their diagnosis of heart diseases, we examine a variety of common deep learning approaches to classify time-series data, such as RNNs and CNNs, to handle large amounts of collected ECG signals. We validate our approaches with the use of two publicly available datasets, the MIT-BIH Arrhythmia Dataset and the PTB Diagnostic ECG Database. We find that with an ensemble of different methods we can correctly identify regular and abnormal ECG signals with up to 99.7% accuracy.

Section 2 describes the aforementioned datasets we use, Section 3 details the various models we examine with an evaluation thereof in Section 4. Finally, Section 5 discusses our findings and Section 6 adds concluding remarks and future work.

2 DATASETS

The MIT-BIH Arrhythmia Dataset (MIT-BIH) contains ~ 110k samples of 48 half-hour recordings of two-channel ECG signals split into 5 classes such as for example normal beat or supraventricular premature beat. These recordings are collected from 47 subjects [7]. The PTB Diagnostic ECG Database (PTBDB) contains ~ 15k samples split into 2 classes, normal and abnormal [3]. The samples in both datasets are preprocessed to a fixed dimension of 187.

3 METHODS

3.1 Task 1: Vanilla Models

3.1.1 Vanilla CNN. We designed our Vanilla CNN architecture with one goal in mind: reducing spatial dimension, and increasing channel dimension as we go deeper. This ensures that we extract all key information from temporal dimension, and distribute these across channels that represent signal characteristics. As this model is aimed to be a baseline, we did not go further than 4 convolutional layers, and did not employ skip connections. After convolutional layers, to reduce the fully connected layer's parameter size, we used global max pooling instead of flattening the feature maps. This completes our baseline CNN model.

3.1.2 Vanilla RNN. An RNN is a natural choice for temporal data as in this task. We simply put the raw signal to RNN architecture, and apply two fully connected layers on top of its last hidden state. The hidden state stores all task-related information, and how it is updated is important for performance. LSTM and GRU cells have advanced gates which prevent the vanishing gradient problem observed in simple RNN cells. However, we implemented our pipeline flexible enough to compare all types.

3.2 Task 2: Additional Models

3.2.1 Bidirectional RNN. A bidirectional RNN processes input in both forward and backward directions. This is problematic for causal applications such as forecasting, but in our case the whole signal is recorded and accessible from both ends. We believe this model is a simple but robust extension to the Vanilla RNN because the start of the signal can now influence the hidden state more in the backward cell. This provides room for finding underlying patterns, should they exist. After passing the raw signal, we simply concatenate the last hidden states of both forward and backward cells and put two fully connected layers on top. This method doubles the hidden parameter size, which can increase model capacity, but might also lead to overfitting. Thus, we tried to find a balance during our experiments.

3.2.2 Residual CNN. Deep neural networks might suffer from vanishing gradients. To overcome this issue, one common technique is to use residual connections as in [5]. Residual connection is simply adding output of a layer to output of another layer. In Residual CNN model, we used four 1D convolutional layers, each with a Leaky ReLU activation, kernel size 3, stride 1 and "same" padding. We normalized each convolutional layer's output with a 1D Batch Normalization [6], therefore we did not use biases in convolutional layers (as suggested in [2]). After the last convolutional layer, we applied a 1D Global Average Pool operation to each channel and applied two consecutive fully connected layers on top of it. First fully connected layer is activated by a Leaky ReLU activation function, and the second one is not activated.

3.2.3 Autoencoder. ECG signals of a heartbeat generally exhibit several wave forms (P, QRS and T). Together with their duration, shape and distances between them, these wave forms can be used to diagnose heart diseases [1]. Using this insight, we seek to learn compressed latent representations of the ECG signals using a convolutional Autoencoder in hopes of capturing information comparable to wave forms, which can then be fed into a classifier to predict the relevant labels.

The encoder uses 3 blocks of 1D convolutions with parametric ReLU (PReLU), batch normalization and max pooling followed by a dense layer mapping into a 30D latent space, a 6:1 compression we have empirically found to work well. The decoder has a dense layer, then 3 blocks of 1D transpose convolutions with PReLU, batch normalizations and upsampling. A final 1D transpose convolution maps back into the input space.

For classification, the ECG signals are encoded and then passed to Scikit-learn's GradientBoostingClassifier [9], which we train using default settings.

3.2.4 1D Inception Net. After analysing the data, we observed that each disease type produces distinct ECG patterns. These patterns vary in temporal length that should be processed by different-sized convolution filters. However, our Vanilla CNN architecture only uses 1D filters of size 3, missing any potential local information. On the other hand, adding more filters leads to a problem: huge increase in channel dimension. Thus, inspired by InceptionNet[10], we decided to control channel dimension with filters of size 1, while using different filter sizes of 1,3, and 5 for feature extraction.

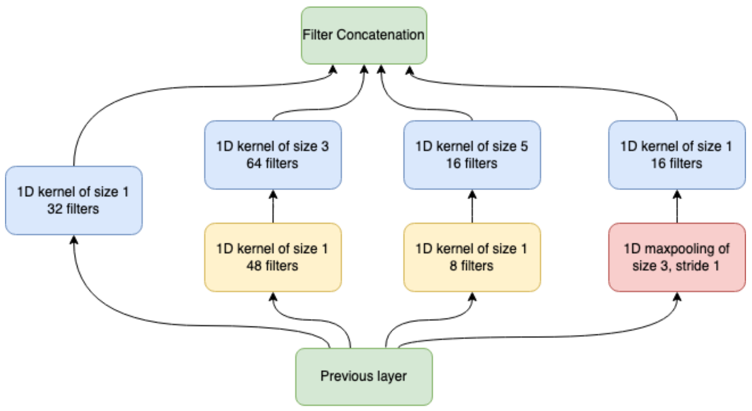

Figure 1: Custom Inception 1D block.

We divided the filters' original channel dimensions by 4 considering the relatively small dataset size, and easier task complexity. Our custom Inception 1D block can be seen in Figure 1. Overall, we applied 16 1D filter of kernel size 7 to the raw ECG signal to increase the initial channel dimension. Then by employing the "max pooling->inception block" pattern 3 times we constructed our final model.

3.2.5 Shared MLP Over RNN. The Vanilla RNN iterates over all 187 timesteps and we put a fully connected layer on top of the last hidden state. Here, we expect the overall signal characteristic to be infused in the hidden state. However, we think that a particular signal might be very indicative about its classification label at some specific timesteps. At this point, waiting until the last timestep might degrade the hidden state's indicative power. Thus, we created a model which uses a shared fully connected layer on top of each hidden state, and learns which timesteps to attend most depending on the signal type. This is done by producing logits by applying the same fully connected layer, and using a softmax to determine importance of each hidden state. Then, we sum up each weighted hidden state and put fully connected layers on top as shown in Figure 2.

3.3 Task 3: Ensemble Models

The different models we described in Section 3 have different inductive biases and hence may struggle with different parts of the datasets. To achieve better overall predictions, we implement ensembles that seek to combine the results of all models.

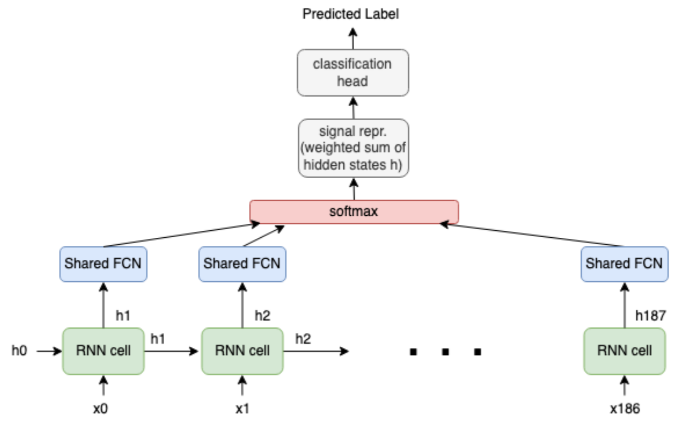

Figure 2: Shared MLP over RNN network.

3.3.1 Majority Voting. As a simple ensemble strategy, we implement majority voting for which we use the predictions of all models described in Section 3. We take the mode of the collection of predicted labels for each sample and use it as a new prediction.

3.3.2 Logistic Regression. While majority voting puts the same weight on each of the ensembled models, we also implement a method which can learn to vary the contribution of each individual model. For this we train logistic regression classifier over the accumulated predictions of our models in Section 3. We use Scikit-learn's logistic regression classifier with default settings apart from using a maximum of 10^5 iterations. We find this approach to outperform simple majority voting as seen in Table 1.

3.4 Task 4: Transfer Learning

Both MIT-BIH and PTBDB datasets are used for similar tasks. However, the PTBDB dataset suffers from data scarcity. This, in theory, reduces generalization ability of any model trained on this dataset. At this point, transfer learning is a reasonable option. A model trained on MIT-BIH dataset has already discovered the potential features needed for the disease classification and can be fine-tuned for the PTBDB dataset. We implemented three fine-tuning strategies. In all of them, we first train both the recurrent and fully connected parts of a neural network on the MIT-BIH dataset. Then we replace the fully connected layer with a reinitialized one, which outputs 1 logit instead of 5, therefore it becomes suitable for the PTBDB dataset.

3.4.1 Strategy 1: Permanent freeze. When the time we replace the fully connected layer, we also freeze the recurrent part of the neural network and keep it frozen during fine-tuning on the PTBDB dataset.

3.4.2 Strategy 2: Never freeze. In this option, the recurrent part of the neural network is never frozen, therefore it is also trained during fine-tuning on the PTBDB dataset.

3.4.3 Strategy 3: Temporary freeze. We freeze the recurrent part of the neural network when the time we replace the fully connected layer, and keep it frozen for 20 epochs during fine-tuning on the PTBDB dataset. At 20th epoch, we start to train the whole network.

3.5 Training Details

We implemented all our deep-learning based models in PyTorch [8] 1.9.0. We used Adam optimizer with learning rate 0.001 without a weight decay. As the learning rate scheduler, we used "ReduceLROnPlateau" with patience of 5 epochs and multiplicative factor of 0.1. Our early stopping criteria had patience of 15 epochs and kept track of the validation loss. We used batch size of 64, and clipped the gradients at norm 5.

4 EVALUATION

For PTBDB dataset, we evaluated the models using accuracy, area under the receiver operating characteristic curve (AUROC) and area under the precision-recall curve (AUPRC) metrics. For MIT-BIH dataset, we evaluated the models using the accuracy metric. AUROC and AUPRC metrics can only be used for binary classification tasks. Therefore, we did not use them for MIT-BIH dataset, which had more than two classes. The scores are provided in Table 1.

5 DISCUSSION

From Table 1, we see that the baseline model has the highest number of parameters, but with clever network design we achieve comparable results with significantly fewer parameters and in case of PTBDB even outperform it using ensembling on top.

Among recurrent cells, LSTMs are better than Simple RNNs, and bidirectionality makes a difference, as we hypothesized. RNN performance is lower for PTBDB than MIT-BIH likely because of data scarcity. Shared MLP over LSTM model meets our expectations about a

	MIT-BIH		PTBDB			
Model Name	#Parameters	Accuracy	#Parameters	Accuracy	AUROC	AUPRC
Baseline CNN [4]	254901	**0.985**	254641	0.995	0.997	0.998
Vanilla CNN	39813	0.95	39681	0.99	0.999	**1.0**
Vanilla SimpleRNN	17413	0.165	16897	0.278	0.502	0.723
Vanilla LSTM	67717	0.946	67201	0.278	0.508	0.726
Bidirectional LSTM	135429	0.974	134401	0.838	0.92	0.968
Residual CNN	39813	0.928	39681	0.969	0.994	0.998
Autoencoder + GBC	**11888**	0.938	**11888**	0.969	0.994	0.998
1D Inception Net	67117	0.974	66857	0.982	0.996	0.997
Shared MLP Over SimpleRNN	25478	0.858	25218	0.544	0.691	0.872
Shared MLP Over LSTM	75782	0.962	75522	0.722	0.5	0.722
Transfer Learning (SimpleRNN, PF)	x	x	16897	0.621	0.726	0.88
Transfer Learning (SimpleRNN, NF)	x	x	16897	0.722	0.507	0.725
Transfer Learning (SimpleRNN, TF)	x	x	16897	0.622	0.724	0.878
Transfer Learning (LSTM, PF)	x	x	67201	0.719	0.795	0.909
Transfer Learning (LSTM, NF)	x	x	67201	0.988	0.996	0.998
Transfer Learning (LSTM, TF)	x	x	67201	0.992	0.999	0.999
Transfer Learning (Bi-LSTM, PF)	x	x	134401	0.864	0.945	0.978
Transfer Learning (Bi-LSTM, NF)	x	x	134401	0.989	0.999	0.999
Transfer Learning (Bi-LSTM, TF)	x	x	134401	0.99	0.998	0.999
Ensemble: Majority Voting	0^*	0.973	0^*	0.995	0.994	0.996
Ensemble: Logistic Regression	55^*	0.979	11^*	**0.997**	**1.0**	**1.0**

Table 1: Performance comparison of models from all 4 tasks, and baselines for both datasets. PF=Permanent Freeze, NF=Never Freeze, TF=Temporary Freeze, GBC=GradientBoostingClassifier. *=only considering ensembling model. Best in bold.

particular timestamp potentially being important. One possible extension to it would be training it with Bi-LSTMs. In addition, 1D Inception Net provides different-sized kernels, which give better coverage than Vanilla CNN, hence higher results.

We showed that these creative models have potential, and increasing their depth or recurrent hidden size might boost the performance further. We also observed that the latent representation of Autoencoder models does not have direct correspondence with the class label. Also, we tried self-attention on RNN hidden states, but this gave low performance and we believe it is because not every timestamp plays a key role in the ECG classification as each word would play in text classification.

On the other hand, transfer learning displays nearly unparalleled results amongst our methods. We see that scenarios where we train a base LSTM with new classification head has better outcomes. It is likely because PTBDB data finds a chance to fine-tune LSTM weights according to its needs. Ensemble learning yield satisfactory results, as the base models were already successful in their predictions. Furthermore, we could have approached this task from feature engineering perspective and tried to extract lots of signal features from 3rd party ECG libraries, then fit an XGBoost classifier.

6 CONCLUSION

Highly accurate ECG classification is possible thanks to deep learning. In addition, wisely crafted neural network models create marginal benefits. In light of these results, via further calibration and data validation, the medical professionals can benefit from this helpful tool.

REFERENCES

[1] Saira Aziz, Sajid Ahmed, and Mohamed-Slim Alouini. 2021. ECG-based machine-learning algorithms for heartbeat classification. *Scientific reports* 11, 1 (2021), 1–14.

[2] Johan Bjorck, Carla Gomes, Bart Selman, and Kilian Q. Weinberger. 2018. Understanding Batch Normalization. https://doi.org/10.48550/ARXIV.1806.02375

[3] R Bousseljot, D Kreiseler, and A Schnabel. 1995. Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet. (1995).

[4] CVxTz. 2019. ECG Heartbeat Classification. https://github.com/CVxTz/ECG_Heartbeat_Classification

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. https://doi.org/10.48550/ARXIV.1512.03385

[6] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://doi.org/10.48550/ARXIV.1502.03167

[7] George B Moody and Roger G Mark. 2001. The impact of the MIT-BIH arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine* 20, 3 (2001), 45–50.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. https://doi.org/10.48550/ARXIV.1409.4842