Machine Learning For Health care Project 2 - Time Series and Representation Learning

Lucia Pancorbo Jona Schulz Fabio Studart

1. Part 1: Supervised Learning on Time Series

1.1. Q1 - Exploratory Data Analysis

The PTB Diagnostic ECG Database is the smaller dataset analyzed in Part 1. In figure 1, we can visualize some examples of the negative and the positive classes. There are 11640 observations for training and 2910 for testing. All time series were padded with zeros to have a fixed size of 187.

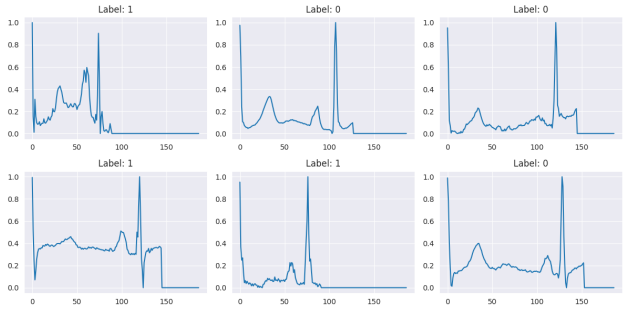

Figure 1. Examples of the two classes of ECG recordings.

The distribution of labels in the training set can be visualized in Figure 2. As there are many more positive labels in this dataset, for the deep learning models, the training was performed using a balanced dataloader for the training set, which gives approximately balanced batches for the computation of gradients. The idea is try to prevent the model to predict all samples as the majority class. For the validation set, all samples were used in each of the 200 training epochs. We decided to save the model parameters in the end of the epoch where it reached the best validation set performance.

To evaluate the models, we decided to visualize the Receiver Operator Characteristic, with its Area Under the Curve (ROC-AUC) as the performance metric. This metric was used as it gives a general estimation of the ability of the model to separate classes, independently of whether a given application requires more sensitivity or specificity. Finally, in summarizing the models, we also decided to compute the F1 score, the balanced accuracy and the Area Under the Curve of the Precision-Recall Curve (PR-AUC). These metrics for all classic and deep learning models will be summarized in tables, respectively, at the end of Q2, and at the of Part 1.

For all deep learning models, we also plotted the training and validation set loss throughout the epochs, which helped us assess bias-variance trade-off, and how fast the models converged. The bottom layers of all deep learning models in this work can be thought of as feature extractors. These features are then used by the equivalent of a multi-layer perceptron to separate the classes. For fair comparison, all our deep learning models have the same head. It consists of two fully connected layers with an ELU activation between them, followed by a sigmoid for the output.

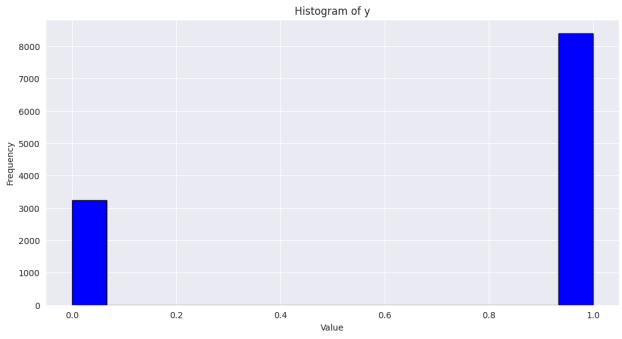

Figure 2. Label distribution in PTB training dataset.

1.2. Q2 - Classic Machine Learning Methods

(1) To benchmark our deep learning models, we use 4 classic/non-deep machine learning models: k-Nearest Neighbours (kNN), Naive Bayes, Random Forest and Multi-Layer Perceptron (MLP), all from the sci-kit learn library (Pedregosa et al., 2011). In figure 3, we can see their performance on the task. We used 100 estimators for the random forest, two hidden layers of size 100 for the MLP and 5 neighbours for the kNN. The Random Forest achieves the best performance in all metrics, while the Naive Bayes performs the worst (See Table 1).

(2) We tried to improve predictive performance by feature engineering. We used the library tsfresh (Christ et al., 2018),

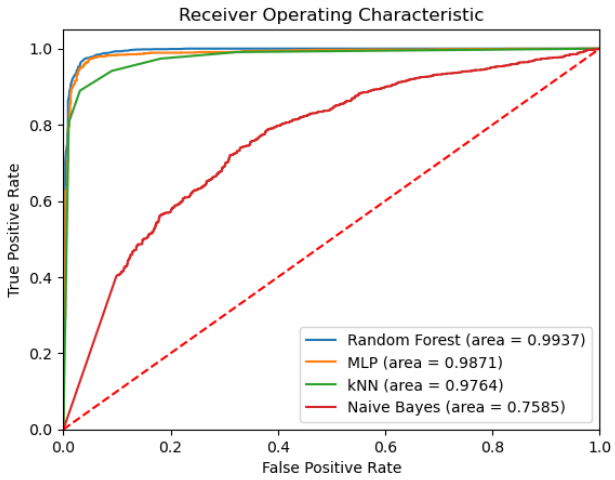

Figure 3. ROC curve with ROC-AUC values for classic machine learning models using as inputs the raw time-series.

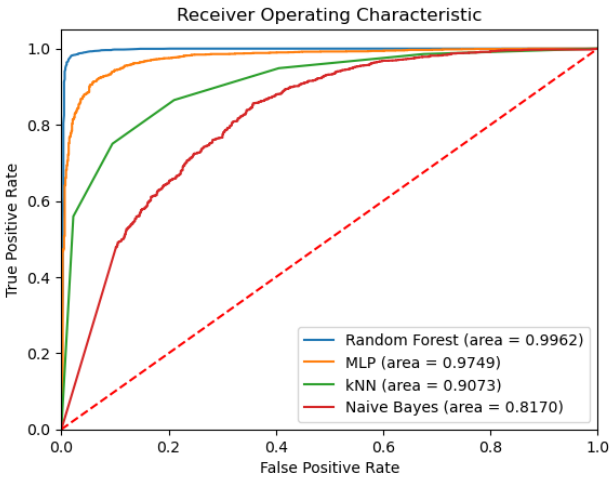

Figure 4. ROC curve with ROC-AUC of classic machine learning models with manual extracted features added to the input.

to extract default features. Then, we removed the NaN values and used feature selection function to filter the most relevant ones for the classification task. We obtained 595 values for each signal, which were concatenated to the time-series. The 3 most relevant ones are Continuous Wavelet Transform coefficients and "ratio_value_number_to_time_series_length" which returns a value of 1 if all values occur only once. Using the same hyperparameters as in the previous question, we can see the results of all classifiers on figure 4. As depicted in Table 1, the Random Forest is still the best of the four, and we can see some improvement in its performance when using feature engineering (similarly for the NB as well). We note that the performance of the MLP and kNN did not improve with the addition of manual features, but decreased slightly, possibly due to a strong increase in dimensionality of the problem with all the features, or also the intrinsic randomness in the training of the MLP.

(3) The choice of classifiers was based on a couple of criteria: First, we used the most simple classifiers, following Occam's Razor: Naive Bayes as the simplest unparametric model, and kNN as a minimally parametrized, but untrained model. Secondly, we chose two models that are considered state-of-the-art classical methods for tabular data: MLPs and Random Forests. Despite not being interpretable, both can model very well non-linear decision boundaries, and we can observe that the performance of Random Forest, both with and without manual features, is similar or better to that of many deep learning models, explored below.

1.3. Q3 - Recurrent Neural Networks

We trained a vanilla Long Short-Term Memory model (LSTM) on the PTB dataset. However, with different degrees of complexity in the architecture, we were not able to get good results initially. The solution consisted of removing the padding with zeros, as they strongly affected training. We hypothesize that the zero values hinder backpropagation through time, as the model head is based on the final hidden state. With many zeros, we believe there are vanishing gradients that stop the model from learning well. Removing the padding, we get a good performance, which can be observed on figure 5.

LSTMs have been recognized as better than Recurrent Neural Networks (RNNs) for remembering long-term data, due to the cell memory, that also allows direct gradient flows through time (Hochreiter & Schmidhuber, 1997) and reduces forgetting of past information. However, many patterns in time series have complex dynamics that depends on initial and final time steps, and the sequential nature of vanilla LSTMs might give more influence to data at the end of the series, or cause "forgetting". To improve on that performance, bidirectional LSTMs were introduced (Schuster & Paliwal, 1997). They can take into account both past and future inputs for prediction, as one network runs backward and their hidden states are used for final computation. We trained this model, and the performance can be seen in figure 6. We note that, in our task, all the information is seen by our model in vanilla LSTMs. Assuming the input and forget gates have been properly trained, looking at the information twice might not improve the test performance. We can observe that bidirectional LSTM do not achieve better performance than vanilla LSTMs, which have a performance on par with most other models.

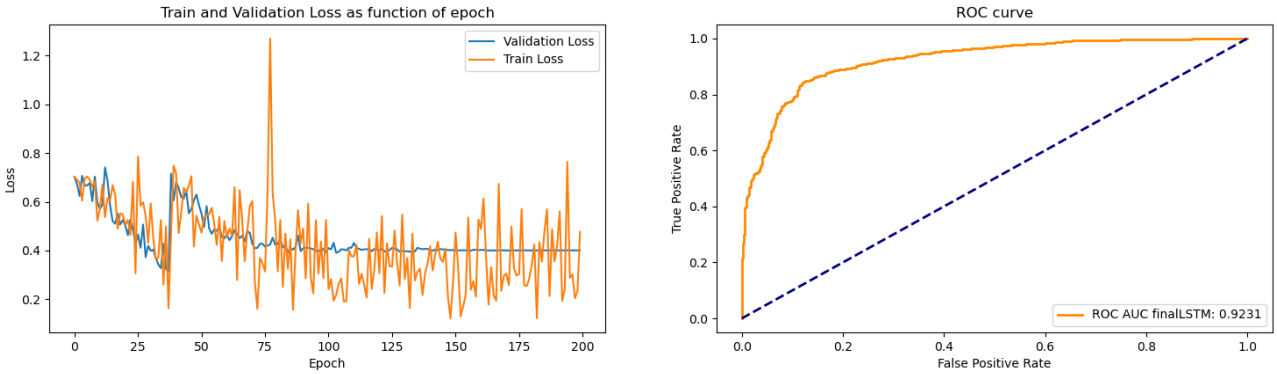

Figure 5. (left) Training and Validation Loss of vanilla LSTM. (right) ROC curve and ROC-AUC of vanilla LSTM.

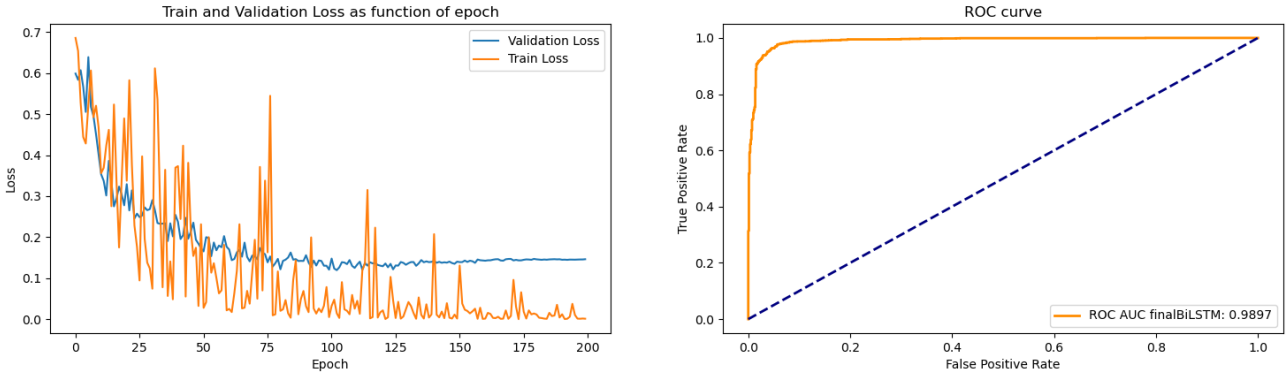

Figure 6. (left) Training and Validation Loss of bidirectional LSTM. (right) ROC curve and ROC-AUC of bidirectional LSTM.

1.4. Q4 - Convolutional Neural Networks

To aggregate information from different points in a time series, in a local manner, convolutional neural networks (CNNs) have proven to be an excellent tool. They serve as automatic feature extractors, when well parameterized, and learn hierarchically more abstract features. They are also translationally equivariant, being able to find the same pattern at different positions. Furthermore, they do not have directionality in their structure, which, for classification tasks, might be an advantage. However, RNNs have some properties that CNNs do not. RNNs can perform online prediction, which is more time efficient. They can also model sequences of variable length, and are inherently good at modelling temporal dynamics due to their recurrence, which is not intrinsic to CNNs.

To compare these architectures, we train two CNN models. For the feature extraction, we use a simple CNN, consisting of multiple blocks of convolution followed by batch normalization and max pooling; we also use a CNN with residual blocks (ResCNN), that adds the input itself to the result of the block, that must have the same dimension.

In figures 7 and 8, respectively, we can see the performance of vanilla CNN and residual CNN. We observe that residual CNNs have a better performance than vanilla CNNs. Besides, we can also see that the residual model reaches its optimal state in less epochs. According to (He et al., 2015), this can be explained because skip connections give a direct pathway for the gradients to flow, without vanishing or explosion caused by, for example, the activation functions. Also, they point out that at each block, the net then needs to learn to perform a transformation minus the identity mapping, instead of the transformation itself, and if we assume that the optimal transformation is closer to an identity than to a zero mapping, this justifies the faster learning of the network.

Having a model with a faster training is a great advantage. An example would be If we want to use our model for the same task, but with a bigger dataset, where we might not have the resources to train the model for the same number of epochs. Looking at the losses across training for the resCNN, we hypothesize that 100 epochs, half of what we used, might be enough to reach near optimal results in

	Balanced Accuracy	**F1-Score**	**ROC AUC Score**	**PR AUC Score**
Raw Time-Series				
Random Forest	0.9660	0.9772	0.9937	0.9973
MLP	0.9621	0.9768	0.9871	0.9939
KNN	0.8978	0.8903	0.9764	0.9911
Naive Bayes	0.7049	0.7926	0.7585	0.8904
Feature Engineering				
Random Forest	**0.9809**	**0.9861**	**0.9962**	**0.9980**
MLP	0.9268	0.9432	0.9749	0.9889
KNN	0.7686	0.7136	0.9073	0.9629
Naive Bayes	0.7493	0.8587	0.8170	0.9152

Table 1. Performance Metrics for Classic Machine Learning Methods with Raw Time-series and with Feature Engineering

a new dataset, assuming the loss landscape has the same complexity.

1.5. Q5 - Attention and Transformers

Finally, the last architecture we experiment with in the PTB dataset classification task is the transformer architecture, using a self-attention mechanism as an encoder. Transformers have been a breakthrough in the deep learning community, allowing a lot of parallel computation with multi-head attention, and the ability of each value in the input representation to be modified by the context (Vaswani et al., 2017). This allows long-term dependencies to be included in the semantic representation of each time point. On the other hand, RNNs always compare the information of the current input with an abstract representation of its hidden state, which has a more modest ability to remember all the past information in a computationally efficient manner.

To be able to visualize the attention maps applied directly to the inputs, we need to have a separate embedding for each time point. We add positional encoding as a second feature in each time, consisting on the normalized position of the input in the sequence. We also applied layer normalization before the attention, as suggested by (Xiong et al., 2020).

The performance of the transformer model can be seen in figure 9. We observe that the model is longer to train, and the validation loss does not show the same behaviour as in the CNNs. We note that there is an inherent trade-off between interpretability and performance. Another more complex model was trained by first projecting all the sequence into a latent representation, and applying self-attention to that representation. It reached similar performance to the CNN models. However, it was not interpretable, because we could not map the latent representation back to each time step. Therefore, to see which time steps is the model paying attention to, we incur in a performance loss.

	Balanced Accuracy	F1-Score	ROC AUC Score	PR AUC Score
Vanilla LSTM	0.8617	0.8937	0.9231	0.9666
Bidirect. LSTM	0.9599	0.9774	0.9897	0.9956
Vanilla CNN	0.9688	0.9748	0.9932	0.9963
Residual CNN	**0.9805**	**0.9873**	**0.9945**	**0.9968**
Transformer	0.8302	0.8523	0.9017	0.9617

Table 2. Metrics values for different Deep Learning Architectures

In figure 10, we plotted, for 3 positive and 3 negative labels, the top 20 time steps that received more attention from other points, on average, in the first attention layer. We can observe that most consistently look at the region after the QRS complex, and the beginning of the T wave, which is visually different in healthy and unhealthy samples. This suggests that the transformer is taking that region more in consideration when classifying the time series.

1.6. Q1-5 - Performance Summary and Model selection

We summarize the performance of our various deep learning methods in table 2. We can observe that in all of our metrics, ResCNN outperforms the other model architectures.

2. Part 2: Transfer and Representation Learning

2.1. Q1 - Supervised Model for Transfer

In this part of the project, we explore transfer learning to improve the performance of our model of choice. We have decided to use the resCNN model due to its performance and speed, which appear promising for training in a larger dataset. The deep learning architecture was modified to an output size of 5 for the arrhythmia classification task, and it was trained on the MIT-BIH dataset. Figure 11 shows the train and validation losses on the training dataset. The validation loss stabilizes around the 100 epochs. Due to class imbalance, we decided to plot Precision-Recall curves besides the ROC curves. Results can bee seen in Figure 12. The AUC values for the ROC curves are very close to 1,

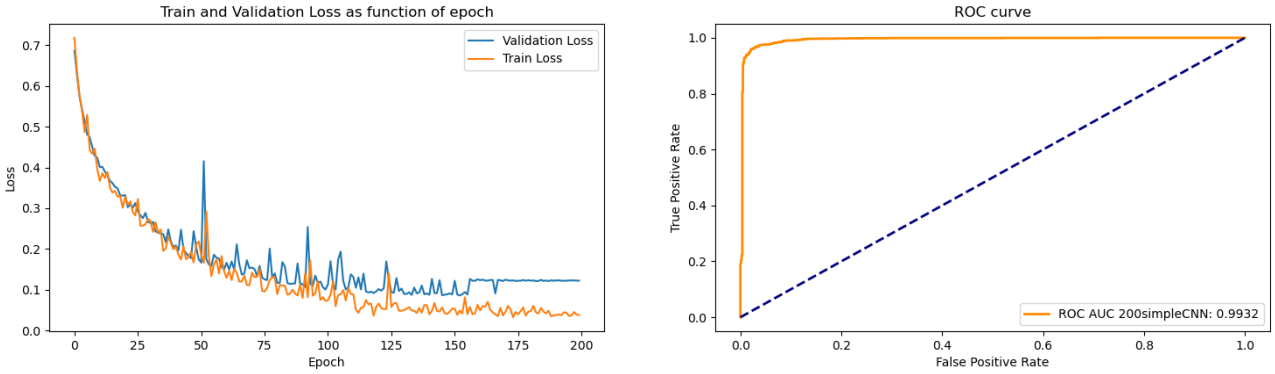

Figure 7. (left) Training and Validation Loss of vanilla CNN. (right) ROC curve and ROC-AUC of vanilla CNN.

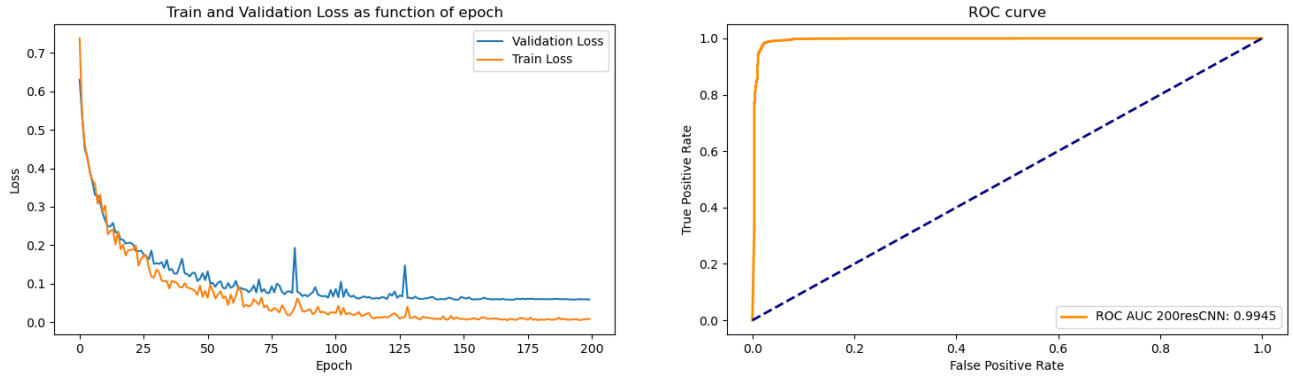

Figure 8. (left) Training and Validation Loss of residual CNN. (right) ROC curve and ROC-AUC of residual CNN.

indicating good performance. Classes 1 and 3 (Categories S and F) have lower AUC values. This means that the model has problems to distinguish those classes, which are the minority categories (See Figure 13). We also report a balanced accuracy of 0.9150 and a f1-score of 0.9536.

Overall, we can conclude that the model shows a great performance on the large dataset. We then proceed to remove the last two fully connected layers to obtain a pre-trained time-series encoder.

2.2. Q2 - Representation Learning Model

Using the same ResCNN architecture from before as an encoder we constructed an autoencoder. The encoder transforms input time series into 128-dimensional latent representations, the decoder consists of three deconvolutional layers to reconstruct the original input. Using a mean-squared error loss function, the autoencoder was trained for 200 epochs in an unsupervised setting. Figure 14 shows some inputs and corresponding reconstructed outputs from the MIT-BIH test dataset.

We then trained a random forest classifier on the representations created by the pre-trained encoder from the MIT-BIH dataset. Figure 20 shows the ROC curve and ROC-AUC score for this classifier for all five labels.

2.3. Q3 - Visualising Learned Representations

2.3.1. Encoder from Q1

We have used the pre-trained encoder based on the MIT-BIH to obtain representations of both the MIT-BIH and PTB datasets. We have performed dimensionality reduction to visualize the results. More specifically, we did PCA (50 components) followed by UMAP because it is usually faster than tSNE on large datasets. Figure 16 shows the representations of the two datasets extracted by the encoder. After trying different UMAP hyperparameter (see Figure 22 and 23), the parameters that provided the best visualization were chosen. To quantify if the points are distributed identically, we used KL divergence for multivariate samples. We used an available piece of code that estimates the KL divergence from samples (Pérez-Cruz, 2008). Since the KL divergence is not symmetric, the displayed values refer to the mean

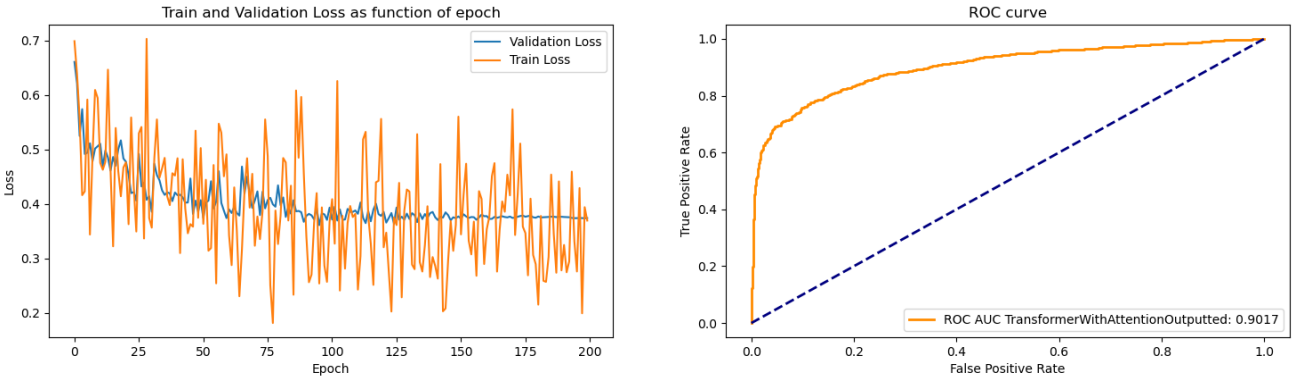

Figure 9. (left) Training and Validation Loss of transformer model. (right) ROC curve and ROC-AUC of transformer model.

	Class 0	Class 1	Class 2	Class 3	Class 4
Class 0	-	**6.1929**	8.2237	7.0922	11.1
Class 1		-	8.9781	8.9451	12.2057
Class 2			-	6.2905	9.6448
Class 3				-	11.1858
Class 4					-

Table 3. Mean KL divergence between representations of different MIT-BIH classes (encoder from Q1). The KL divergence in both directions was computed and averaged.

	Class 0	Class 1	Class 2	Class 3	Class 4
Class 0	-	3.18	4.48	4.01	7.45
Class 1		-	4.48	6.01	8.17
Class 2			-	4.47	6.88
Class 3				-	**8.3**
Class 4					-

Table 4. Mean KL divergence between representations of different MIT-BIH classes (encoder from Q2). The KL divergence in both directions was computed and averaged

between the divergence in both directions.

The KL divergence between MIT-BIH and PTB datasets is 8.8628, meaning that they are not identically distributed. Conversely, the KL divergence between the two labels of the PTB 3.9302, meaning they are more similarly distributed as can be observed in the image. The comparison between classes for the MIT-BIH dataset is shown in Table 3. In this case there is no much similarity in the distributions. The lowest value is between class 0 and class 1, and in Figure 16 we see that all data points with label 1 overlap with those with labels 0. Interesting, the ResCNN achieved the worse performance in the class 1 of MIT-BIH dataset (See Figure 12), which can be explained by the lack of distinction with class 0 in the representations.

2.3.2. Encoder from Q2

We visualized the representations generated by the encoder trained in Q2 in the same manner. The resulting UMAP visualizations are shown in Figure 17. The resulting KL-divergences between the distributions of MIT-BIH representations of different labels are shown in Table 4. The largest divergences occur between the distributions of samples with label 4 and the remaining samples which is reflected in the UMAP visualizations by the distinct cluster of samples with label 4. The KL-divergence between the PTB and MIT-BIH dataset samples is 6.11.

2.4. Q4 - Finetuning Strategies

In this section, we will use the pre-trained encoders to improve the classification on the PTB dataset. We follow two different strategies Classic ML method and fully connected output layers.

2.4.1. Encoder from Q1

Classic ML method: The PTB dataset is fed to the pre-trained ResCNN encoder to extract the embeddings. Then, we trained a Random Forest (100 estimators) on the training embeddings. We chose this classifier because it was the best performing one in Section 1.2. We evaluated it on the test embeddings and results are shown in Figure 18. The AUC value is very close to 1 for ROC and PR curve, being 0.9954 and 0.9977. It achieves a balance accuracy of 0.9753 and a f1-score of 0.9840.

Fully Connected output layers: For this approach we added two fully connected layers to the encoder for the binary classification task. We then did finetuning at different levels. In strategy A, we froze the encoder and trained only the output layers. In strategy B, we trained the entire model. In strategy C, we trained in two separate stages, first the output layers and then the entire model. ROC curves for the three strategies are shown in Figure 19, and Table 5 contains the evaluation metrics.

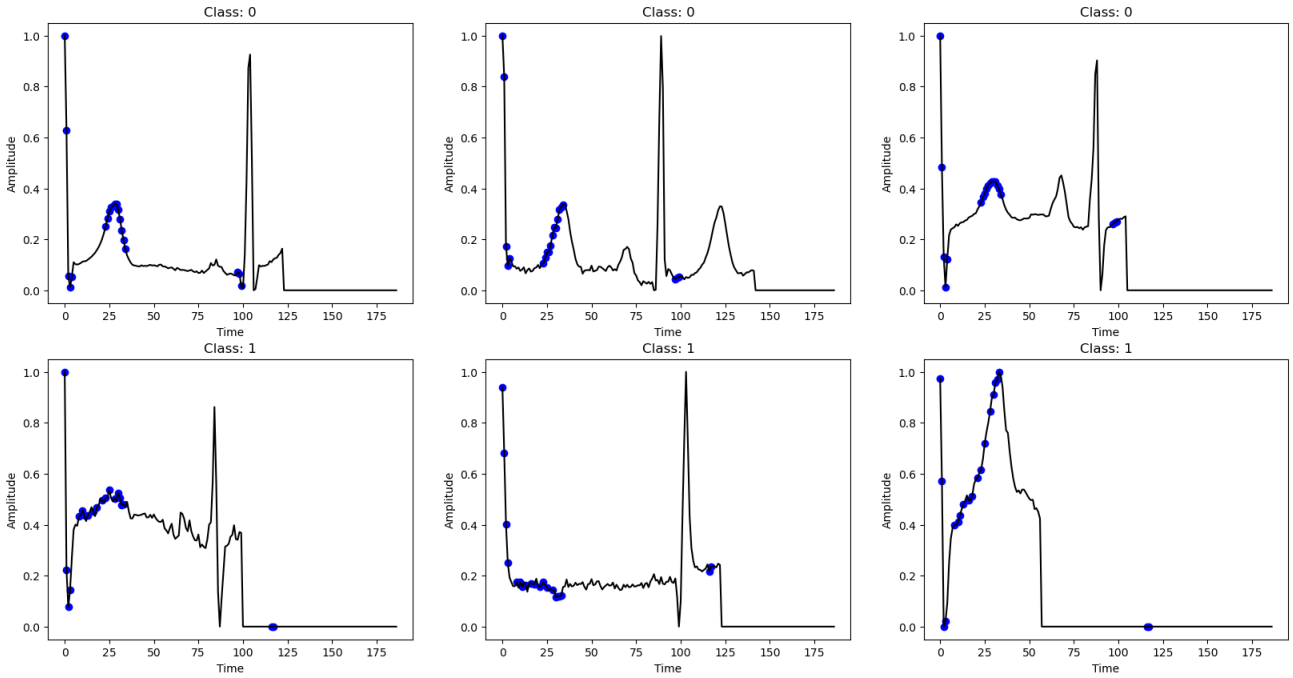

Figure 10. Top 20 points with higher received average attention, for input sequences of different labels.

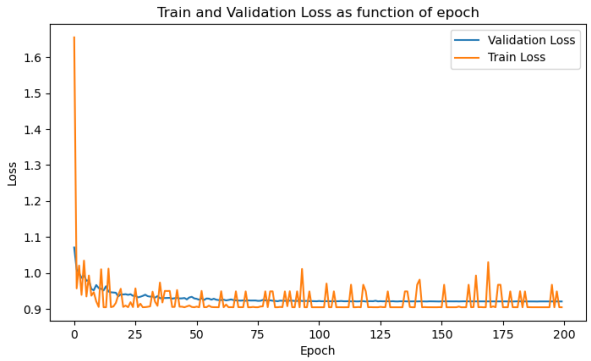

Figure 11. Training and Validation Loss of the ResCNN model on the MIT-BIH dataset.

For transfer learning, the strategy that gave us the best result is B, training the whole model (encoder + output layers) on the PTB dataset. This could be because there are more trainable parameters, which allows the adaptation of the model to features that are specific to the dataset. Additionally, we saw in Figure 16 that the pre-trained encoder on the MIT-BIH does not separate the two classes of the PTB dataset. For this reason, it is better to re-train it on the dataset.

2.4.2. Encoder from Q2

Classic ML method: Equivalently, the previously outlined fine-tuning strategies for transfer to the PTB dataset were applied to the encoder trained in Q2. A random forest classifier was trained once again on the PTB representations generated by the pre-trained encoder from Q2. The resulting ROC curve and ROC-AUC score are shown in Figure 20. The random forest classifier achieves a ROC-AUC score of 0.982 which is worse than what was achieved for the encoder from Q1.

Fully Connected output layers: Further, the three fine-tuning strategies of freezing the encoder, training encoder and output layers as well as the two-stage training strategy were tested. Once again, a two-layer MLP was added to the output of the encoder to act as a classifier. The resulting ROC curves and ROC-AUC scores are shown in Figure 21. The balanced accuracies, F1, ROC-AUC and PR-AUC scores are shown in the bottom half of Table 5. Here, strategy C works best and actually yields the overall best ROC-AUC and PR-AUC scores of all models trained on the PTB dataset throughout this project. The lowest performance is achieved by strategy A which implies that fine-tuning the encoder is beneficial in the case of this model.

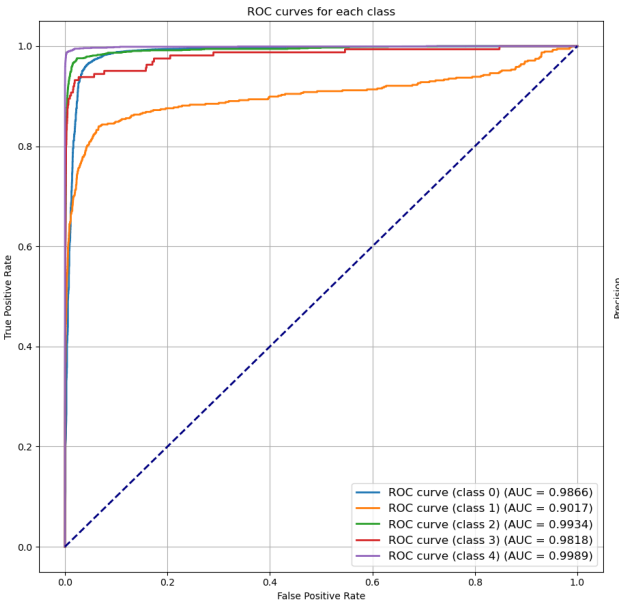

Figure 12. Evaluation of ResCNN on the MIT-BIH test dataset. ROC curve and ROC-AUC values for each class using a one-vs-all approach.

2.4.3. SUMMARY

As evident by our results, both the transfer learning and the representation learning approach yield better performing classifiers on the PTB dataset than simply training a classifier from scratch on this dataset. Fine-tuning of an encoder pre-trained on the larger MIT-BIH dataset results in a measurable improvement in classification performance. The performance is also improved with respect to the Deep Learning architectures from Part 1 (Compare Tables 2 and 5).

The finetuning stragies that worked best are Strategy B and C (Table 5). Strategy A, involving freezing the encoder, gave the worse performance. This means that the encoders pretrained on the MIT-BIH datasets need to be finetuned on the PTB dataset.

There seems to be no different between transfer and representation learning. This means that both representations learnt by the encoder are equally good, as they both achieve to classify the PTB classes correctly. Note that the best strategy depends on the metric considered, but they still get very similar values.

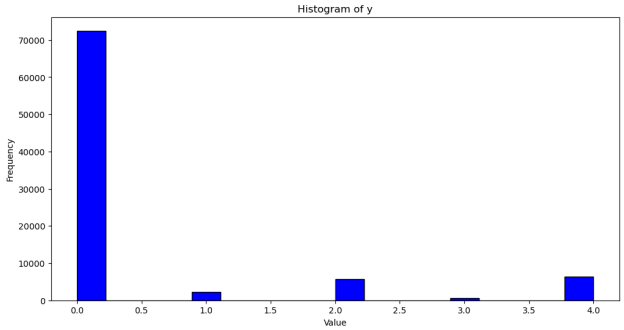

Figure 13. Label distribution for the MIT-BIH dataset where the labels correspond to categories 'N': 0, 'S': 1, 'V': 2, 'F': 3, 'Q': 4

	Balanced Accuracy	**F1-Score**	**ROC AUC**	**PR AUC**
Transfer Learning				
Strategy A	0.9822	0.9883	0.9956	0.9976
Strategy B	**0.9897**	**0.9924**	0.9964	0.998
Strategy C	0.9682	0.9799	0.9915	0.9959
Representation Learning				
Strategy A	0.9569	0.96	0.9881	0.9943
Strategy B	0.9803	0.9842	0.9965	0.998
Strategy C	0.9888	0.9894	**0.9972**	**0.9987**

Table 5. Performance metrics following different fine-tuning strategies. Comparison Transfer Learning and Representation Learning.

3. Part 3: General Questions

Q1 - There are many machine learning settings where classic methods are still competitive with deep learning architectures. Have you observed this in this project? Why is this (not) the case?

In our case, the random forest, known for its state-of-the-art performance in tabular data, has achieved a very strong performance, especially with manually extracted features. Its leveraging of bootstrapping and random feature selection allows almost independent classifiers, which, on average, create a very strong predictor. It has similar or better performance than many of our deep learning approaches to the task, even when leveraging transfer learning or representation learning with a bigger dataset. However, it is not, in the case, the best. In part 1, only with the PTB dataset, the ResCNN model achieves better F1 score than the Random

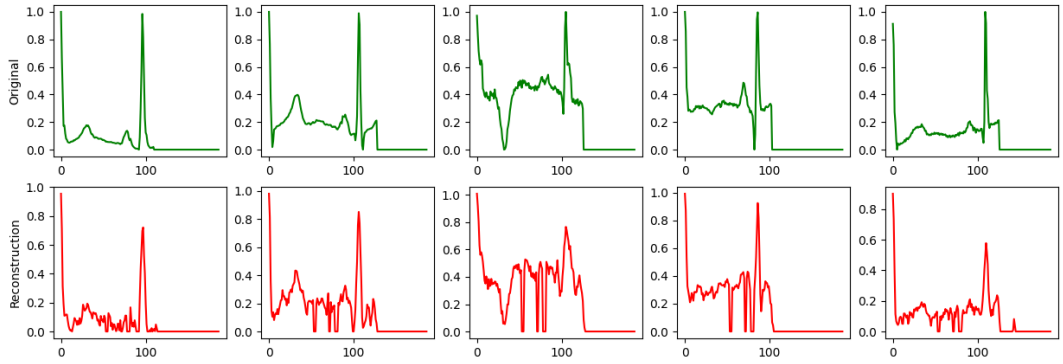

Figure 14. Autoencoder inputs and reconstructions (MIT-BIH test dataset).

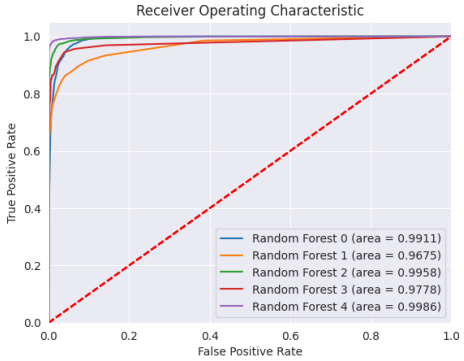

Figure 15. ROC curve and ROC-AUC scores per class for a random forest classifier trained on the MIT-BIH latent representations created by a pre-trained encoder (obtained from the autoencoder trained in the unsupervised setting).

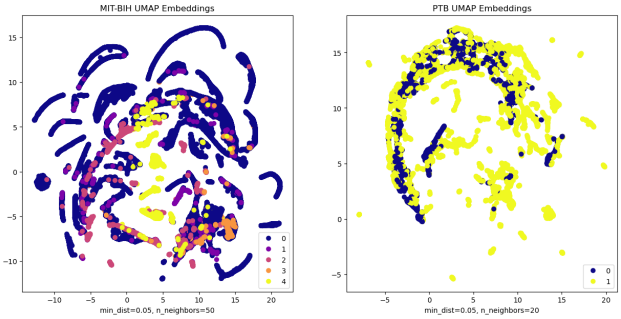

Figure 16. Visualization of representations extracted on both datasets by a ResCNN encoder pre-trained on the MIT-BIH dataset after dimensionality reduction.

Forest, despite having worse values on other metrics. Furthermore, we observe that, leveraging the MIT-BIH dataset, at least one of the training strategies, for both representation and transfer learning, achieve better performance than the classic model.

Q2 - When modeling time series using deep learning architectures, when might you want a causal/unidirectional architecture? Give an example of a task.

In an online setting, or in forecasting, where future time points are not available, a unidirectional architecture is necessary. This could be predicting the weather, or estimating human poses from a video without having access to future frames.

Q3 - Can you think of an attention-related bottleneck regarding very (very) long time series? Conceptually, which deep methods from above are more suitable for such long time series?

An attention-based architecture scales quadratically in complexity as a function of the input sequence length. This is due to the size of the Query, Key and Value matrixes. Therefore, for very long sequences it might become infeasible to use attention. Other methods such as RNNs or CNNs are more suitable in this case. The model do do not scale in number of parameters with the size of the sequence.

What are some challenges in using self-supervised representation learning? What difficulties have you observed in your approach? Can you think of additional ones?

Learning representations in a self-supervised setting does not necessarily yield suitable representations for a different

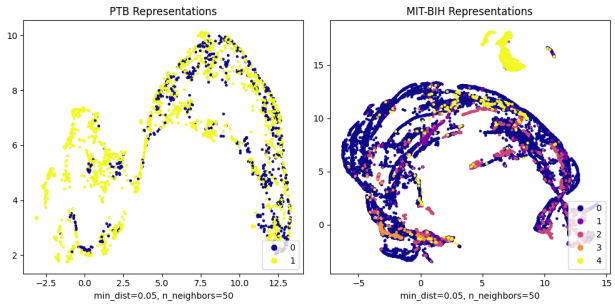

Figure 17. UMAP visualization of MIT-BIH and PTB test dataset representations generated by the encoder trained in Q2.

task. While we did observe performance improvements by using self-supervised representation learning, the UMAP visualizations we obtained showed no clear separation of samples with different labels in most cases. Ideally, representations used for classification should be nicely separated in the embedding space. Achieving this by means of self-supervised training is quite difficult though as the label information is not available during training.

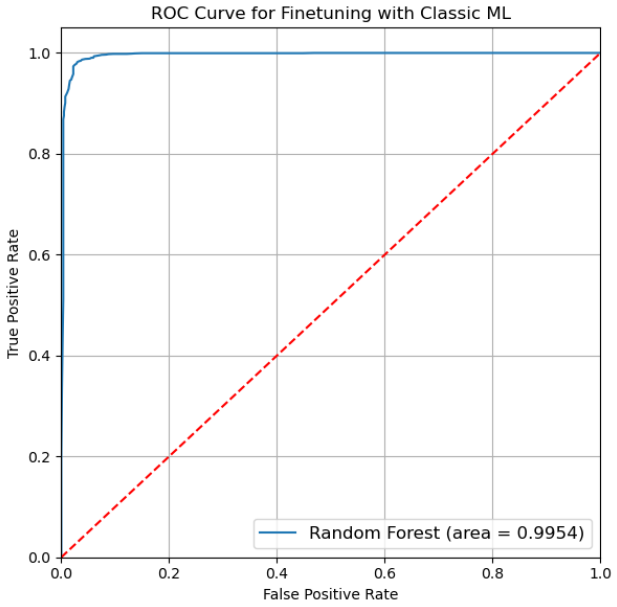

Figure 18. ROC Curve for transfer learning with Classic ML strategy (PTB Dataset).

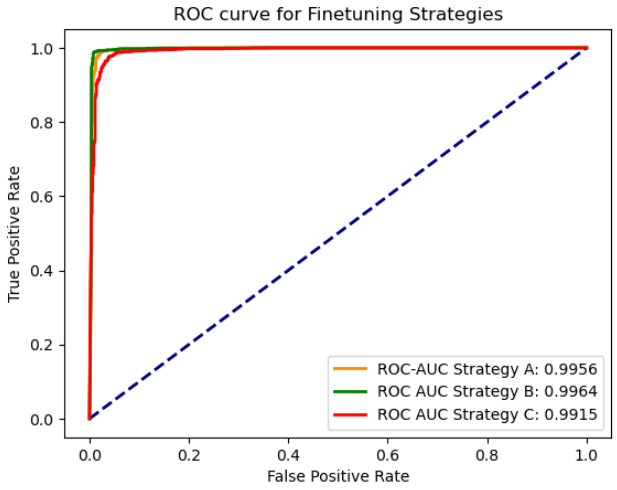

Figure 19. Evaluation of transfer learning with Fully Connected output layers strategies (PTB Dataset). **Strategy A:** Train the output layers only; **Strategy B:** Train the entire model; **Strategy C:** Training of components in two different stages

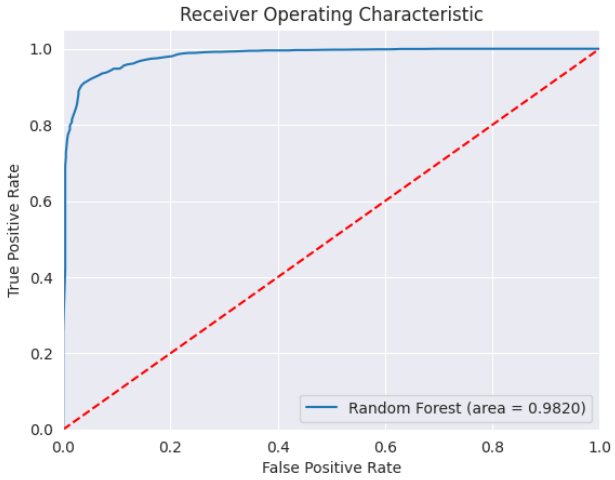

Figure 20. Random forest classifier trained on PTB representations obtained from the encoder trained in Q2.

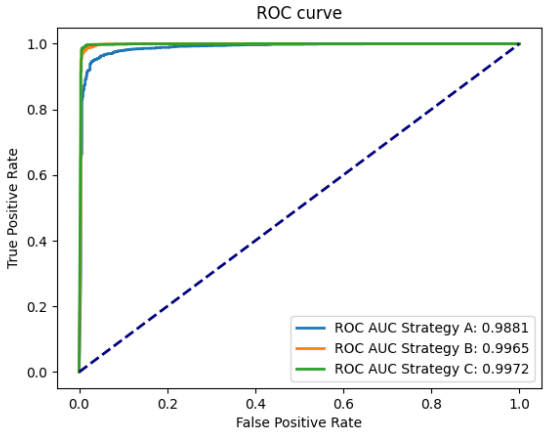

Figure 21. Different fine-tuning strategies on the PTB dataset for the encoder trained in Q2.

References

Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W. Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package). *Neurocomputing*, 307:72–77, 2018. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.03.067. URL https://www.sciencedirect.com/science/article/pii/S0925231218304843.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

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

Pérez-Cruz, F. Kullback-leibler divergence estimation of continuous distributions. In *2008 IEEE international symposium on information theory*, pp. 1666–1670. IEEE, 2008.

Schuster, M. and Paliwal, K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45 (11):2673–2681, 1997. doi: 10.1109/78.650093.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Xiong, R., Yang, Y., He, D., Zheng, K., Shuxin, Z., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T.-Y. On layer normalization in the transformer architecture, 02 2020.

4. Appendix

In this section we added the results when testing different hyperparameters for the UMAP dimensionality reduction. Note the 50 principal components are used to speed up the process.

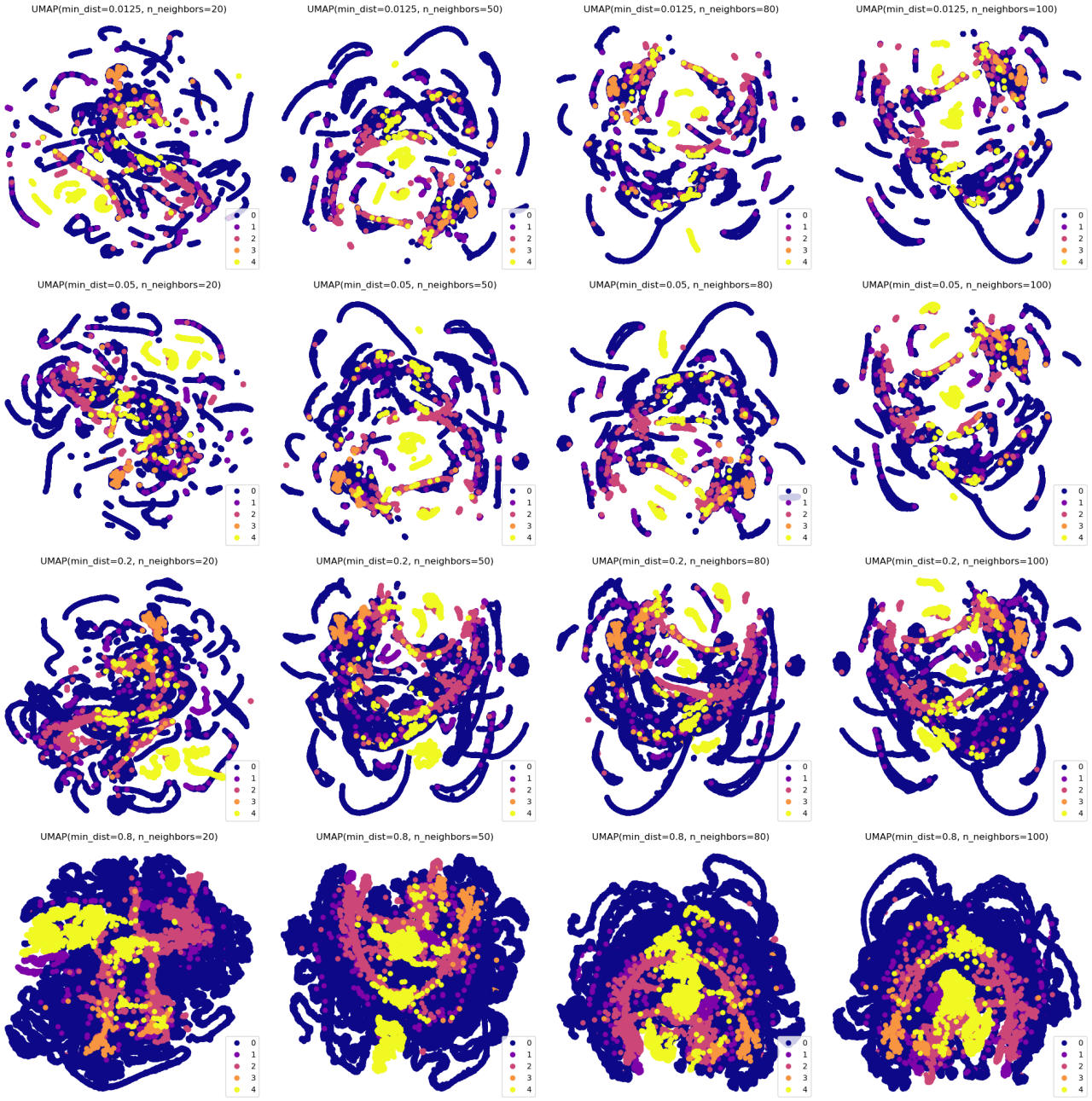

Figure 22. UMAP embeddings obtained from the Transfer Learning Encoder on the MIT-BIH dataset

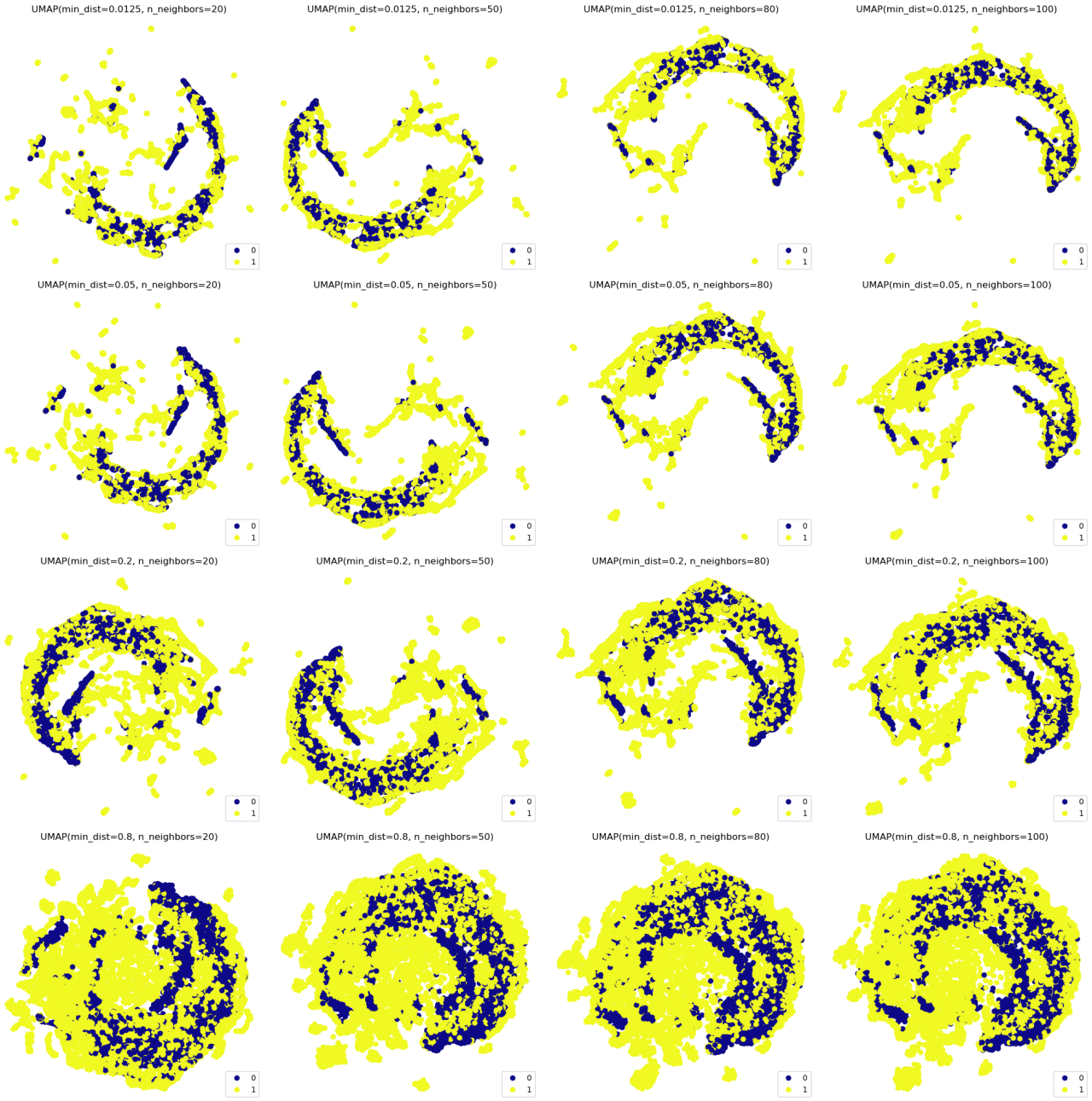

Figure 23. UMAP embeddings obtained from the Transfer Learning Encoder on the PTBDB dataset

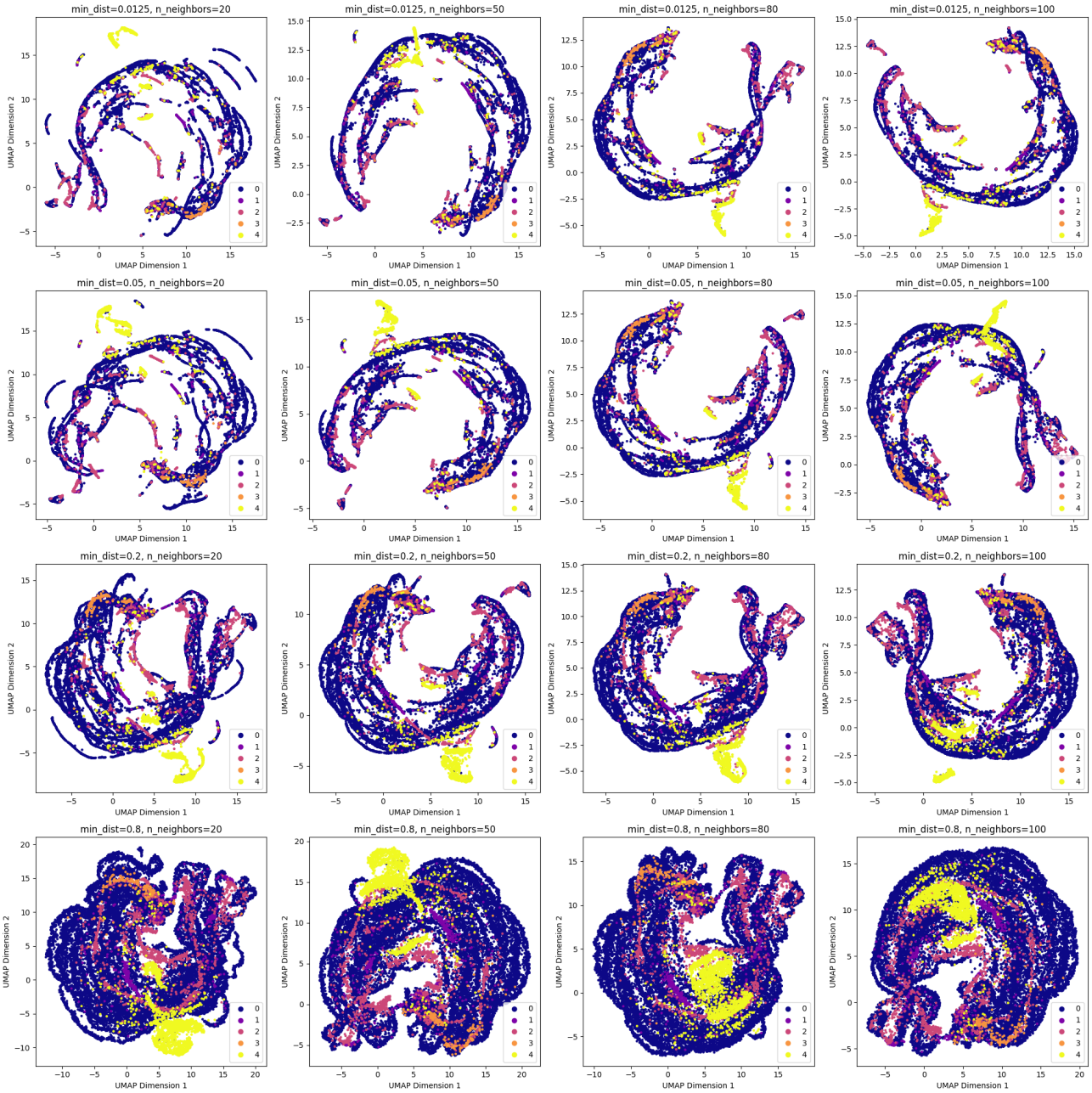

Figure 24. UMAP embeddings obtained from the representation learning encoder on the MIT-BIH dataset

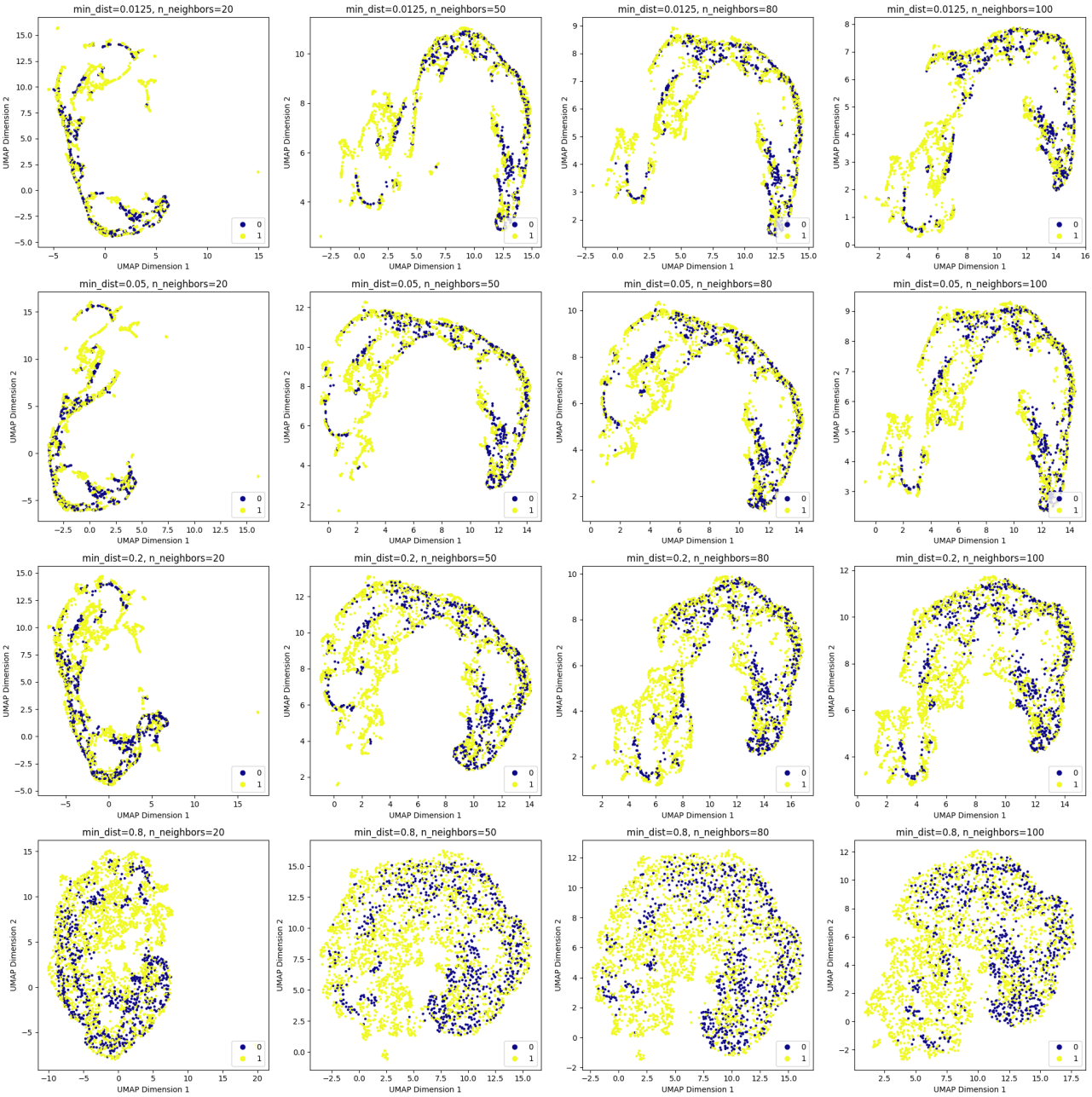

Figure 25. UMAP embeddings obtained from the representation learning encoder on the PTBDB dataset