ML4H Project 2

Sophia Houhamdi, Eva Sarlin, Lorenzo Tarricone

May 2024

1 Supervised Learning for Time Series

1.1 Exploratory data analysis

PTB dataset is a labelled dataset for binary classification, classes are imbalanced (3237 zeroes vs 8404 ones in the train and 809 vs 2102 in the test set).

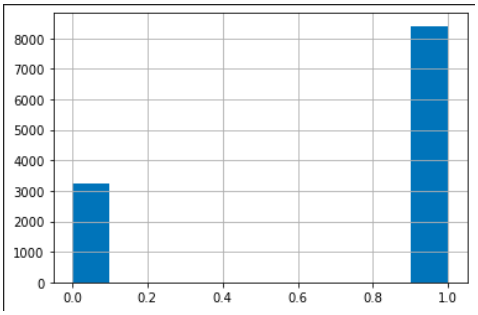


Figure 1: Class labels count in the PTB dataset.

All ECGs have the same length (187 timepoints) and have some common features: the timeserie starts on the descending part of the peak, and then one period is observed with the peak being at a varying location in the sequence accross the dataset. The sequence always ends with a padding of zeroes to ensure that all ECGS have the same length (preprocessing).

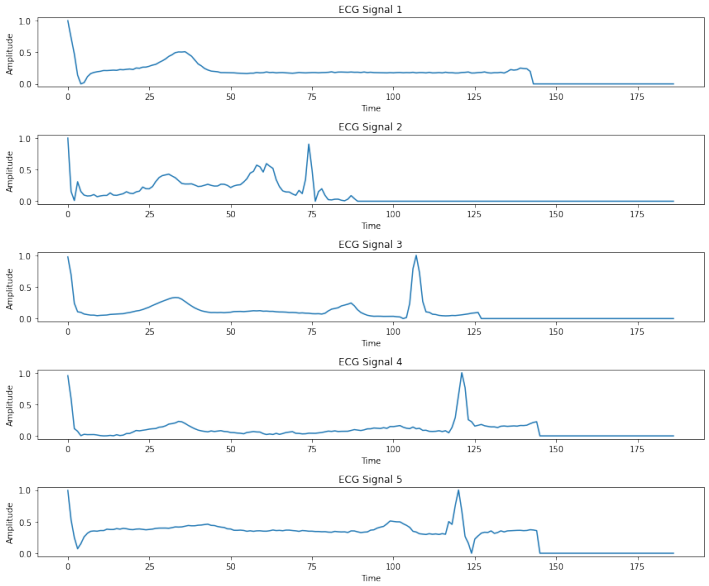
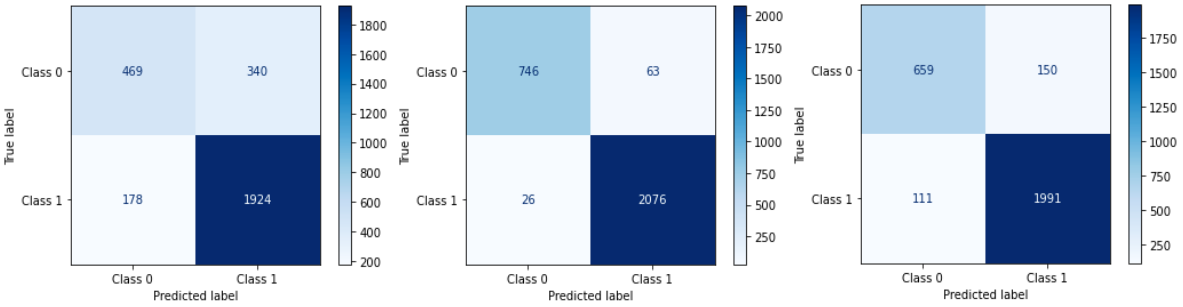


Figure 2: ECG samples from the PTB dataset.

For this binary classification task, appropriate metrics are: accuracy, F1 score, and Area Under the Receiver Operating Characteristic Curve (AUC ROC). In the rest of this project, we will always plot the confusion matrix and provide the accuracy and the f1 score.

1.2 Classic Machine Learning Methods

For this part, we first trained three different classical machine learning methods on the raw PTB time series: logistic regression, random forest and support vector machine (SVM). Confusion matrices and test set performance (accuracy and f1 score) are reported in Figure 3 and Table 1. We can see that random forest outperforms the logistic regression and support vector machine.



(a) Confusion matrix from logistic regression model.

(b) Confusion matrix from random forest model.

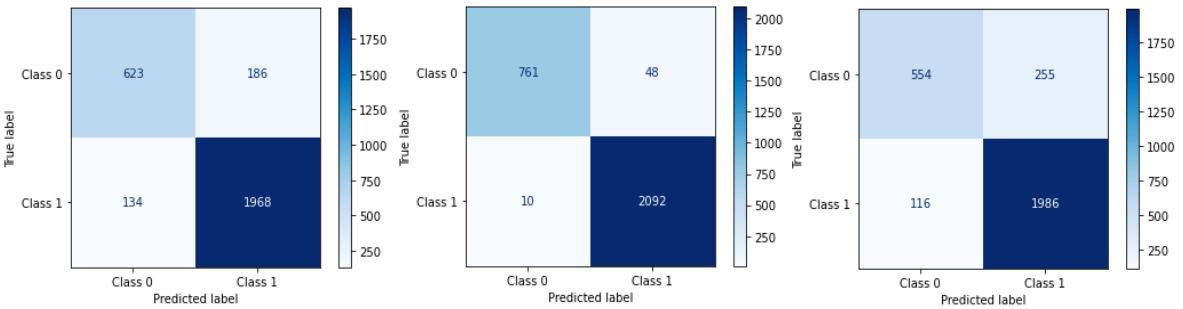
(c) Confusion matrix from support vector machine model.

Figure 3: Confusion matrices following different classical machine learning methods trained on raw time-series from the PTB dataset.

	Logistic Regression	Random Forest	Support Vector Machine
Accuracy	0.8221	0.9694	0.9103
F1 Score	0.8813	0.9790	0.9385

Table 1: Performance Metrics on the raw time-series PTB dataset.

Then we used the library tsfresh to extract some additional features in the timeseries. In order to do so, we used tsfresh functions extract_features and select_features to extract and only keep features that are relevant for this classification task (features related to the fast fourier transform of the signals, like the coefficients, fourier entropy, spectral density coefficients, aggregated linear trend ...). We used them as additional features to the raw time-series and retrained the same three classical models on the new dataset enriched with these new signal processing-based features. We report test set performance in Table 2 and confusion matrices are plotted in Figure 4. Adding these new features improved accuracy for logistic regression and random forest but not for SVM, surprisingly.



(a) Confusion matrix from logistic regression model.

(b) Confusion matrix from random forest model.

(c) Confusion matrix from support vector machine model.

Figure 4: Confusion matrices following different classical machine learning methods trained on PTB dataset enriched with signal processing-based features extracted with tsfresh.

	Logistic Regression	Random Forest	Support Vector Machine
Accuracy	0.8901	0.9801	0.8726
F1 Score	0.9248	0.9863	0.9146

Table 2: Performance Metrics on PTB dataset enriched with signal processing-based features extracted with tsfresh.

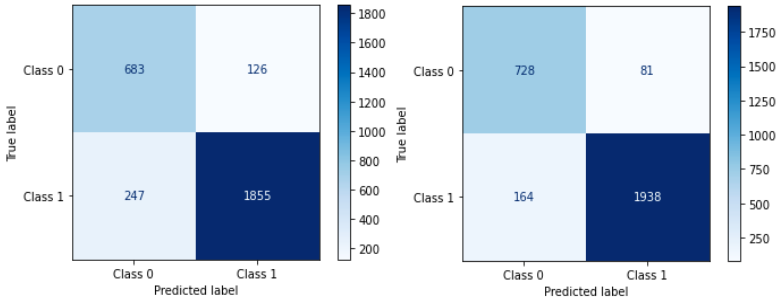
Every classifier has pros and cons :

- Logistic regression : simple and fast, provides probability estimates for each class but it can perform badly for complex non-linear patterns like ECGs.
- Random forest : able to capture complex non linear relationships, robust to noise and outliers, but lacks interpretability and is more computationnally expensive compared to regression.
- Support Vector Machine : can model non-linear decision boundaries using kernel tricks, effective for high-dimensional data but lacks interpretability.

1.3 Recurrent Neural Network

	unidirectional LSTM	bidirectional LSTM
Accuracy	0.8719	0.9158
F1 Score	0.87	0.918

Table 3: Performance metrics of the uni- and bidirectional LSTM models on the PTB test set.



(a) Confusion matrix for unidirectional LSTM.

(b) Confusion matrix for bidirectional LSTM.

Figure 5: Confusion matrices for two types of recurrent neural networks.

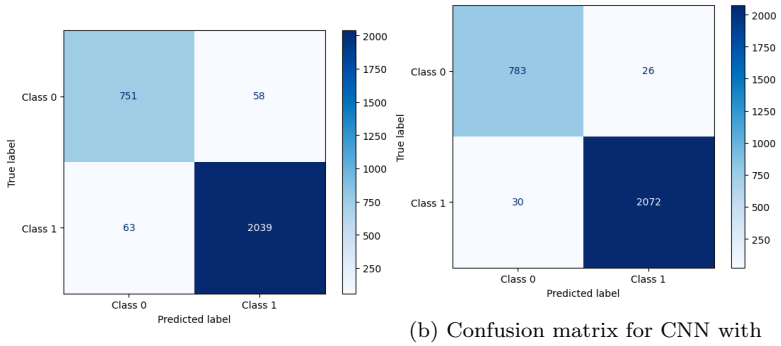
Using Pytorch class `nn.LSTM` and playing with the parameters 'bidirectional', we built two LSTM, one unidirectional, one biderectional. We can see that Bi-LSTM archives better performance compared to "vanilla" LSTM. This is probably because, in the architecture of the former, the processing of the information is done (as the name suggests) in two directions. In particular, it consists of two separate LSTM layers - one processing the input sequence in the forward direction, and another processing it in the reverse direction. The hidden states from these two layers are then combined at each time step, allowing the model to incorporate context from both past and future time steps.

1.4 Convolutional Neural Networks

When performing classification task on timeseries, RNN and CNN have different relative advantages. RNN are designed to handle sequential data and thus are natural candidates to capture temporal dependencies in the data, using their internal memory. However without long-term memory and skip connection, they may struggle with long sequences and exhibit vanishing or exploding gradients, although LSTMs represent a solution to these problems. Additionnally, LSTMs are more prone to

overfitting than CNN and they also have a lot of hyperparameters to tune, which might be time consuming and computationally expensive. On the other hand, CNNs are rather designed to extract local features in the data and thus can be used to detect trends and seasonality in the data by applying convolution filters over the timeserie, but they do not handle time dependencies beyond the receptive field of the filters (fixed by the kernel size and the CNN depth); they are initially more suited for image classification rather than timeseries classification. Moreever, in contrast to RNNs, CNNs require a fixed size input and can't handle a dataset with timeseries of varying sequence length, requiring padding and thus heavier pre-processing steps.

We first built a rather deep vanilla CNN with five convolution blocks, each involving two convolution layers and ReLu activations and a max pool. Then one skip connection was added in each convolution block to built a second CNN, the recurrent CNN. The architecture for this second model is illustrated in figure 7 taken from the paper 'ECG Heartbeat Classification: A Deep Transferable Representation' by Mohammad Kachuee et al. However, training a model with so many layers can require a substantive amount of time, therefore, anticipating the contrastive learning in the second task, we also trained a simple CNN with only two convolutions, so that we can reuse the same architecture in different questions and compare all results.



(a) Confusion matrix for vanilla CNN.

(b) Confusion matrix for CNN with residual blocks.

Figure 6: Confusion matrices for two types of CNN.

	Vanilla CNN	Residual Blocks CNN
Accuracy	0.96	0.98
F1 Score	0.9712	0.9867

Table 4: Performance metrics of a standard CNN and a CNN with skipped connections on the PTB test set.

From our results, we see that for the same architecture the addition of a skipped connection after every convolution block (five in total) results in a smaller value of the loss (0.014 vs 0.112) and better classification metrics. The reason is probably because the presence of residual connections facilitates the training of deep CNN giving a faster way for the error to backpropagate through the network. Interestingly enough, we have noticed this effect to be less evident for the shallower CNN that we have trained for the contrastive encoder, confirming this hypothesis.

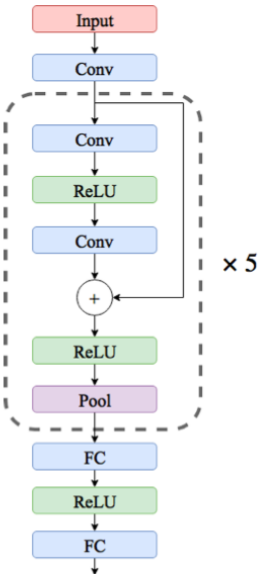


Figure 7: Architecture of the CNN with residual skip connections.

1.5 Attention and Transformers

We implemented a custom transformer encoder block starting from the modules `nn.TransformerEncoderLayer` and `nn.TransformerEncoder` by creating a custom encoder that would also return the attention weights. The positional encoding was taken from a tutorial on the official Pytorch website. We stacked two such encoder blocks before the fully connected head with hidden dimensions 100 and 4 attention heads. After a training of 10 epochs, we had an 82.79% accuracy on the test set.

In our case, we noticed a smoother training loss curve for the Transformer when compared to the LSTM architecture. Moreover, as shown below, the attention weights of the Transformer allow for better interpretability of the model.

We then plotted the returned attention weights from the attention block. From our experiments, we noticed that a lot of attention is put on the peaks of the time series when the peaks are over a certain height. We suppose that the position of the peak in the time series might be indicative of the class of the patient.

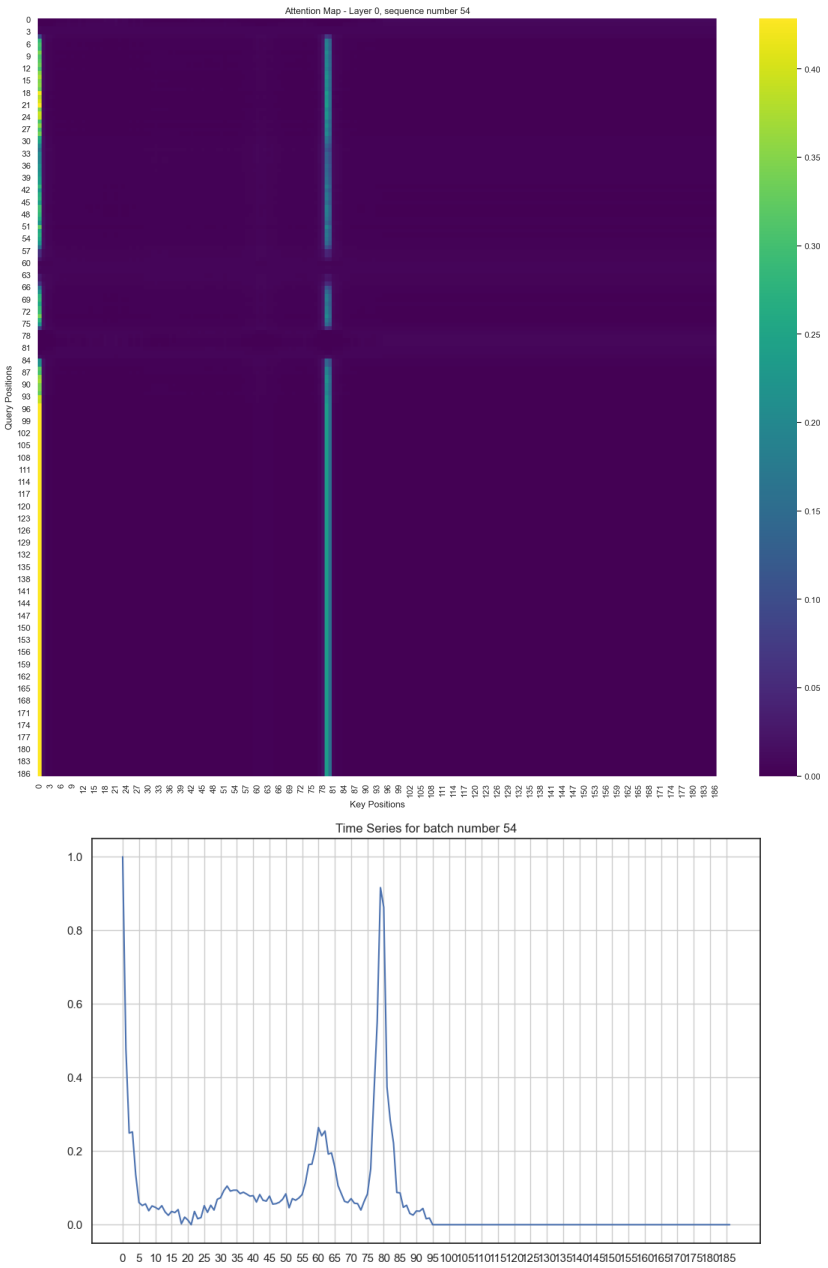


Figure 8: Top: attention heatmap, bottom: relative time series

2 Transfer and Representation Learning

2.1 Supervised Model for transfer

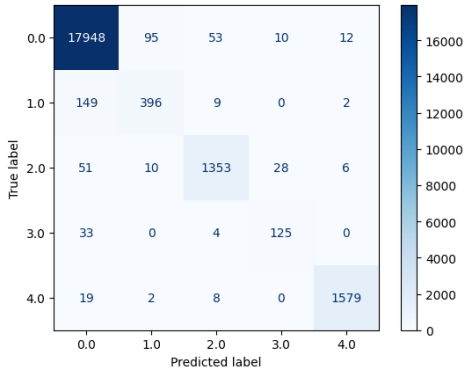


Figure 9: Confusion matrix of the (small) CNN classifier on MIT test set.

For this first transfer learning task we used the same CNN architecture that we will use for generating contrastive embedding below. As discussed in the general questions, this has a smaller architecture with respect to the CNN used in the first part. We chose this model because in the previous part it showed faster and more stable training.

From the reported confusion matrix one can obtain some relevant metrics for this (multiclass) classification task, such as accuracy and F1 score. The first metric is chosen because it's easy to interpret and therefore easier to share for example with a clinician, the F1 score is instead chosen to overcome problems related to class imbalance when accounting just for accuracy, because it takes into account both precision and recall.

2.2 Representation Learning Model

For the representation learning model, we chose to build an encoder based on prediction contrast.
Contrastive learning is an unsupervised method that consists in learning a new embedding of the data by contrasting between positive and negative samples : specifically, positive samples should have similar representations, while negative samples have a different one. For the time-series data at hand, we chose to sample the negative and positive samples as next time steps to predict. To perform this kind of prediction, we fine tuned an encoder by building an auto-regressive model on top of it, which is supposed to learn to classify accurately the next time steps, among all the classes defined by the negative and positive samples, according to the context captured so far.

As suggested by the assignment, we used the InfoNCE approach described in the paper [Oord, Li, and Vinyals, "Representation Learning with Contrastive Predictive Coding." https://arxiv.org/abs/1807.03748] which uses a probabilistic contrastive loss that aims at finding not the optimal posterior distribution of the data to predict given the context, but rather maximize the mutual information between the context and the data (this approach is supposed to better capture global features).
We implemented our model based on the Github reposit of Cheng-I Lai [https://github.com/jefflai108/Contrastive-Predictive-Coding-PyTorch.git].
We used a simple 2 layer CNN for the encoder (with batch normalization and ReLU activation function at each layer), providing embeddings with a downsampling factor of 4 and 64 channels, as for the encoder of Q1. For the Autoregressive model used to monitor the pretraining of our encoder we chose a GRU RNN with 256 dimensionnal hidden state to output the context. We chose a window size of 50 (so that we are in the window of the actual signal without padding), 7 time steps to predict, and a batch size of 8 (which is the number of samples used for classification during the contrastive learning process). We obtained good results when training for 20 epochs on the MIT-BIH dataset to get the

embeddings and then performed a random forest classifier using the labels provided, with an overall accuracy of 97%. The results are summarized in Figure 11.

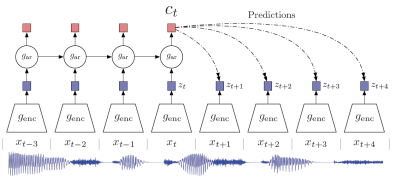


Figure 10: Overview of Contrastive Predictive Coding Architecture, extracted from the InfoNCE paper

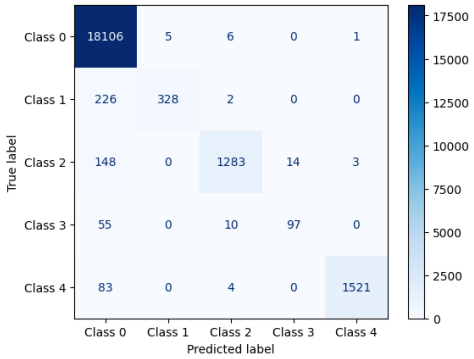
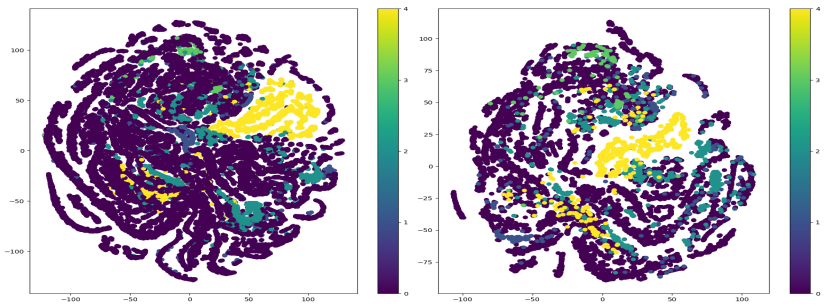


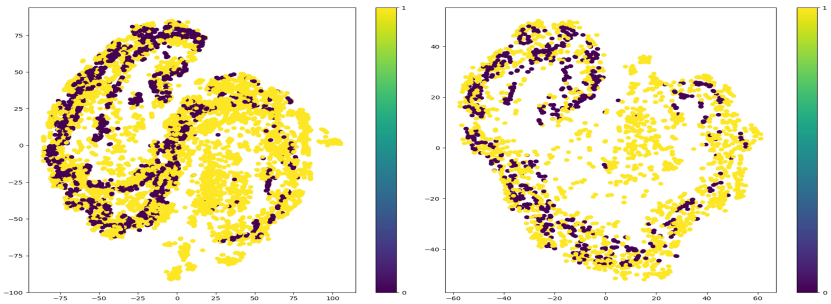
Figure 11: Confusion matrix of the random forest classifier on MIT test set representations learnt with the constrastive loss encoder.

2.3 Visualizing Learned Representations



(a) CPC learnt representation for MIT train set.

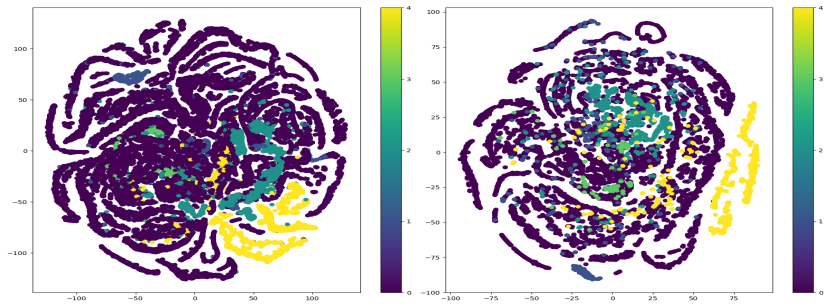
(b) CPC learnt representation for MIT test set.



(c) CPC learnt representation for PTB train set.

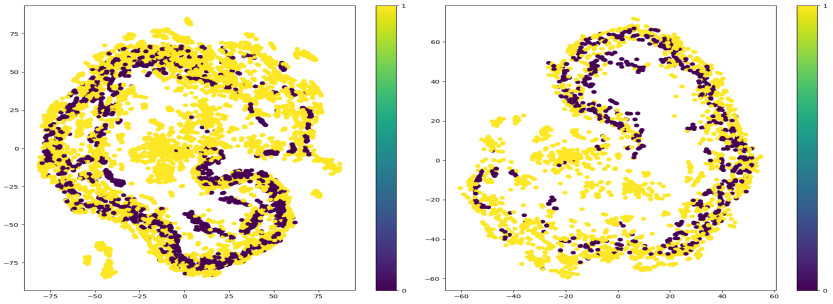
(d) CPC learnt representation for PTB test set.

Figure 12: t-SNE visualization of the constrastive prediction code encode.



(a) CNN learnt representation for MIT train set.

(b) CNN learnt representation for MIT test set.



(c) CNN learnt representation for PTB train set.

(d) CNN learnt representation for PTB test set.

Figure 13: t-SNE visualization of representations learnt by the CNN encoder.

From the images obtained we have can clearly see how the shape of the manifolds of datapoints in the embedding space are consistent across the two encoding strategies. This might ensure us that meaningful representation are leaned independently of the encoding strategy. This shows that sometimes self-supervised learning methods (therefore not needing labels) can give embeddings qualitatively comparable to embedding learned in a supervised way
The two datasets are distributed in different ways (independently of the t-SNE parameter we have tried) and for both datasets we see a different distribution of the classes across the manifolds. For the MIT dataset we can clearly identify regions (such as the yellow one corresponding to label 4) in one specific region of space. This difference in distribution is less apparent for the PTB dataset.

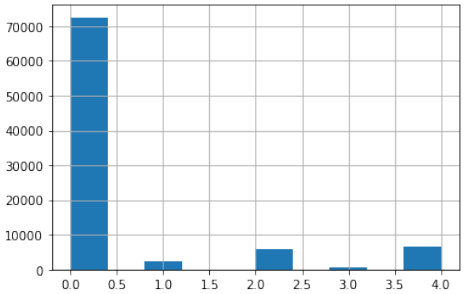


Figure 14: Class labels count in the MIT dataset.

2.4 Finetuning Strategies

1. Classic ML method:
We trained a Random Forest (RF) on the two different embeddings of the PTB dataset provided by the encoders in Q1 and Q2. The results are reported on the tables below. We can see that the transfer learning embedding (Q1) performs slightly better than the representation learning one (Q2), but both performs very well with accuracies and F1-scores around 97%. Such hierarchy of embeddings is expected as transfer learning builds an encoder with label of the classification task

at hand, whereas representation learning builds the encoder on a surrogate task which is contrastive predictive classification. It is thus remarkable that they perform almost equivalently. Compared to the performance observed in Part 1 of the project while training exclusively on the PTB dataset, we can see that we obtain comparable results for the three strategies. With transfer learning embedding performing slightly better then raw data (consistent with the fact that the embedding training is done a larger data set and that embeddings might capture more global features) almost equivalent to the RF trained with raw and additional features. Representation learning performs the least well but only with 0.1%.

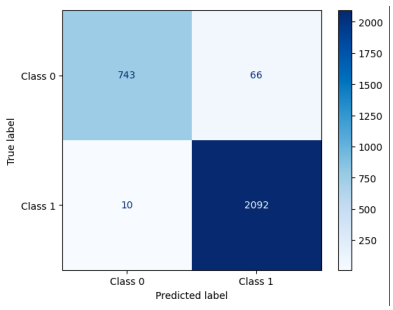


Figure 15: Confusion matrix of the random forest classifier on PTB test set representations learnt with the pre-trained constrastive loss encoder.

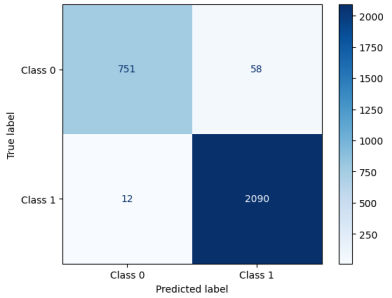


Figure 16: Confusion matrix of the random forest classifier on PTB test set representations learnt with the pre-trained CNN encoder.

2. MLP output layers:

We then used an MLP as classifier by finetuning the whole models using three different strategies as specified in the assignment. The results are reported in the tables below. As we can see, for both embeddings, strategy C is the one which performs best with the highest accuracies and F1-scores (even compared to RF); then strategy B follows and finally strategy A. These results for strategy C are the ones expected as it relies on a fine tuning strategy that combines both knowledge from the MIT data set (as it uses its weights for initialization), and the specificity of PTB dataset when retraining the whole model on it. It is surprising that strategy B performs better that strategy A as the idea of transfer learning is that we train on a larger data set so that we learn a better representation. This could be explained by the fact that the PTB dataset size (11641 ECGs) and nature of data are already good enough for the task at hand (as showed in part 1), so it is more relevant to completely fine-tune on the PTB dataset with the desired labels.

We can also notice that the transfer learning approach performs best than the representation learning one (accross all strategies), which is what we expected because, as said above when comparing with RF, the first relies on embeddings with label-oriented training while the other one is self-supervised (so it trains on a further task than the one that we need).

(a) Confusion matrix following strategy A.

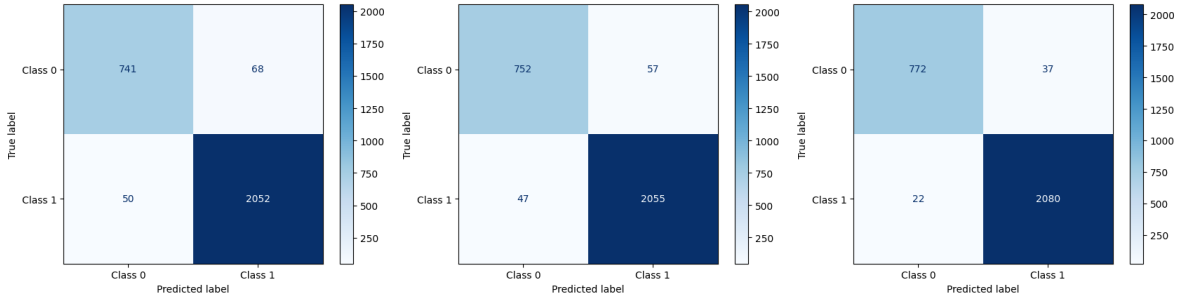
(b) Confusion matrix following strategy B.

(c) Confusion matrix following strategy C.

Figure 17: Confusion matrices following different finetuning strategies on the PTB dataset when using the contrastive loss encoder pretrained on the MIT dataset and following different finetuning strategies for classification.

	Random Forest	Strategy A	Strategy B	Strategy C
Accuracy	0.9668	0.957	0.959	0.968
F1 Score	0.9739	0.971	0.972	0.978

Table 5: Performance Metrics for the different finetuning strategies when doing transfer learning from the CPC encoder pretrained on MIT to PTB dataset.



(a) Confusion matrix following strategy A.

(b) Confusion matrix following strategy B.

(c) Confusion matrix following strategy C.

Figure 18: Confusion matrices following different finetuning strategies on PTB dataset when using the CNN pretrained on the MIT dataset.

	RandomForest	Strategy A	Strategy B	Strategy C
Accuracy	0.9760	0.959	0.964	0.980
F1 Score	0.9835	0.972	0.975	0.986

Table 6: Performance Metrics for the different finetuning strategies when doing transfer learning from the CNN encoder pretrained on MIT to PTB dataset.

3 General Questions

3.1 Classic methods vs Deep Learning

According to the results obtained, we can see how classic machine learning methods (e.g. Random Forests and Support Vector Machine) often obtain better results in terms of our metrics when compared to more complex and modern architecture (e.g. LSTM and Transformers).

We think this might be due to factors such as the reduced size of the training dataset (the 11641 samples of the PTB dataset are nothing compared to the size used to train transformers) or the relative simplicity of the single datapoints themselves (LSTM and transformers are very flexible models and able to understand long-distance relationships of long time/tokens series. Here each time series was consisting of just 187 values).

Last but not least the addition of engineered features from the signal processing domain (such as Fourier coefficients) would most surely give sufficient global features of the time-series (as we are dealing with periodic function such as the ECG signal) and using RNNs can be an overkill.

3.2 Causal/Unidirectional architecture for time series

When modelling time series using deep learning architectures, you might want to use a causal or unidirectional architecture in scenarios where you need to make predictions based solely on past observations, without any information from future time steps. This is particularly relevant in real-time forecasting applications, such as demand forecasting when doing inventory optimization because during the training you can ensure that the predictions at a given time step are based solely on past observations and not influenced by future data points.

3.3 Attention bottleneck for time series

The primary bottleneck with attention mechanisms in very long time series is their inability to efficiently handle long sequences due to the quadratic growth in memory and computational requirements. This issue arises because the self-attention mechanism needs to compute attention scores for all pairs of input tokens, leading to a significant increase in resource consumption as the sequence length grows. For this reason, it would probably be better to use models such as LSTM or CNN, with the first one being probably the best in handling long series with recurrent patterns such as the ones that can probably arise in long ECG reads.

3.4 Challenges in using Self-Supervised Learning

Self Supervised Learning (SSL) methods require a lot of data to be effective in learning good representations because they need to be able to "shape" the embedding space so that it can learn a good representation for all possible future data that will be embedded. This need to process a large amount of data will also lead to large computing times. In this assignment, we had to create and then use a much shallower CNN for the contrastive embedding with respect to the one originally created in Part 1 Question 4, because of the computing time otherwise required.

We can assume that the data from different patients are independent of one another and that the manifold from which they come is stable in time (i.e. the ECG done on one patient one year is not that different from the one made on another patient the following year). This assumption is not true for example when deploying a time-series-based model when the distribution from which the data come from changes in time. One example of such a scenario could be a time series forecasting made over a very long time where the effects of unprecedented historical events (therefore not present in the training data used to shape the embeddings) come in.