ECG Classification using a hybrid CNN-Transformer model

Kyle Seaman
Georgia Institute of Technology

Elisabeth Gutierrez-Zaheer
Georgia Institute of Technology

I. Introduction

Cardiovascular diseases (CVD) are the leading cause of death worldwide. A major challenge for clinicians is the timely identification and diagnosis of patients to provide adequate treatment [1]. Artificial Intelligence has been at the forefront of extracting patterns from raw data, ECG (electrocardiogram) signals, without human intervention, offering highly accurate interpretations that enable rapid clinical analysis and diagnoses [1]. Additionally, AI reduces data processing time, providing real-time insights and saving clinicians significant time [1]. Using both raw signal data (temporal dependent data) and its time-frequency representation (i.e. spectrograms), the hybrid CNN-Transformer architecture presented in [2] classifies ECGs signals for Arrhythmia detection and distinguishes between normal and abnormal cases in the PTB dataset described in [3].

II. Scope of reproducibility

The project aims to reproduce the hybrid CNN-Transformer architecture model described in the paper [2]. This architecture includes two modules: one processes the raw ECG signal using a 1D CNN-Transformer, while the other processes the spectrogram representation using a 2D CNN. A third module includes a fusion mechanism that combines the previous two modules to create a hybrid model which can be used for signal classification as seen in Fig. 15. The fusion module takes the single feature space from each of the previous two modules and creates a multi-feature hybrid model. The paper presents two experiments to be reproduced. The first experiment compares the two individual models against the multi-feature hybrid model. This experiment explores how combining the feature space of both models can increase performance as opposed to each of the individual model's feature space. The second experiment compares different fusion methods within the fusion module. The different fusion methods analyzed are late fusion and three variations of intermediate fusion. This experiment demonstrates the differences between fusing the output classification scores of each model (late fusion) and fusing the feature representation of each model (intermediate fusion). Both experiments aim to demonstrate the effectiveness of a model based on multi-features extraction and learnable attention weights to do classification on both raw and transformed signal data.

The datasets used are the Physionet MIT-BIH Arrhythmia dataset and the Physionet PTB Diagnostic dataset, each containing ECG signals. The HITS dataset, which contains information on cerebral emboli, is a private dataset and is not available on Physionet or any other public data repository. As a result, this dataset is not used.

III. Methodology

A. Model Descriptions

1) 1D CNN-Transformer

This model processes the raw signal to create classification scores. This model architecture includes two encoders, as shown in Fig. 16. The 1D CNN Encoder consists of six one-dimensional convolutions with a kernel size of three and a stride of one [2]. After each convolution, the Leaky ReLU activation function is applied allowing a small gradient for negative values. Following the final four convolutions, a max pooling with a kernel size of two is applied [2]. Padding is not used in the first and final four final convolutions. The resulting output consists of 128 output channels which are processed to generate signal embeddings [2]. Each embedding has a class token prepended for classification by the transformer. These embeddings are added to sinusoidal positional embeddings, as implemented in [4], to retain sequence information when processed by the transformer encoder [2].

The encoder has eight attention heads and a feed-forward with dimension of 64 to transform the output from the self-attention layer [2]. It includes eight layers of the self-attention and feed forward layer, each followed by layer normalization. After the transformer encoder generates an output, the class token is passed through layer normalization and then a linear layer to produce the class scores [2]. The Noam optimizer is applied during training for the 1D CNN-Transformer encoder as implemented in the following paper [4].

The Noam Optimizer increases the learning rate linearly for the first warm up steps in the training steps and decreases proportionally to the inverse square root of the step number scaled by inverse square root of the dimensionality of the model [5]. Table I shows the values used when training the model.

Model	β_1	β_2	**Warm-up steps**
1D CNN Transformer	0.9	0.999	4000

TABLE I: Noam Optimization Parameters [2]

2) 2D CNN

This model processes the time-frequency representation of the signal. This representation is in the form of a spectrogram image. The spectrogram image is constructed using n_{fft} = 32, $n_{overlap}$ = 4, and a Blackman window [2]. Since the data samples are recorded over a 1.5 second period, the sampling rate equals 125 [2]. The produced spectrogram is converted to a 128 x 128 resolution to avoid memory constraints. A sample of the spectrogram transformation is shown in Fig. 1.

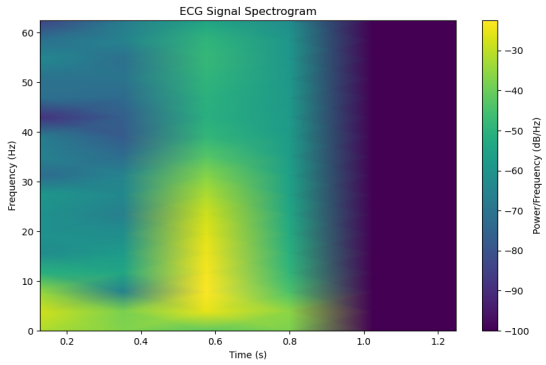


Fig. 1: ECG spectrogram transformation

The 2D CNN model takes the spectrogram image as input. The input size is (3, 128, 128), which represents the 3 color channels along with the image dimensions. As shown in Fig. 17, the 2D CNN model architecture consists of 4 2D convolution layers, each followed by a leaky ReLU activation function, a batch normalization layer, and a max pooling layer. Each layer has 8, 16, 32, and 64 filters, respectively, with size (3, 3) and stride 1 [2]. The max pooling layer has a kernel size (2, 2), padding of 0, and stride 2. After the last convolution layer, there is 1 fully connected layer followed by a dropout of rate 0.2. An Adamax optimizer is used during training.

3) Late Fusion

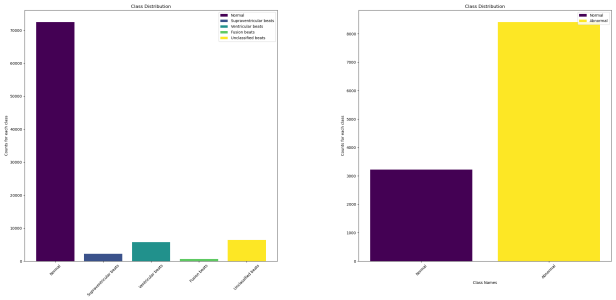
The Late Fusion model creates a hybrid of the 2D CNN model and the 1D CNN-Transformer model by combining the output scores of each model. By fusing the output score of each model, the combined output score has multi-feature representation. For each model, there is a set of learnable weights. An element-wise multiplication is performed on each of the model's outputs and one of the learnable weight vectors. Then the product of each element-wise multiplication is summed to the get the final combined output score of the fused model. The learnable weights allows the Late Fusion Model to learn the significance of each of the model's feature space and apply them respectively to minimize loss.

4) Intermediate Fusion

The Intermediate Fusion creates a hybrid model of the two individual models, similar to the Late Fusion model. However, this fusion method combines each model's feature vector rather than their output score. In the 2D CNN, the feature vector is represented by the output of the last max pool layer, right before it is passed to the fully connected layers. In the 1D CNN-Transformer, the feature vector is represented by the output of the Encoder, right before any layer normalization and fully connected layer. Both of these feature vectors are passed through a fully connected layer in order to get them in a space of equal dimension. There are three methods used for combining the feature vectors: concatenation, sum, and weighted sum. The concatenation method concatenates both feature vectors to get a combined feature representation. The sum method sums both feature vectors. The weighted sum method multiplies each feature vector by a weight and then sums them. In the weighted sum method, the weights are learnable weights, allowing the model to learn feature significance between each of the individual models. For all 3 methods, the combined feature representation is then passed into a fully connected layer to obtain the final output score.

B. Data Descriptions

The paper utilizes the **ECG Heartbeat Categorization Dataset** from Kaggle [3], which combines data from Physionet MIT-BIH Arrhythmia dataset and the Physionet PTB Diagnostic dataset. The ECG signals are cropped, downsampled and zero-padded to a fixed length of 188. The MIT-BIH Arrhythmia dataset includes 109,446 ECG samples recorded at 125 HZ, categorized into five categories: normal, supraventricular beats, ventricular beats, fusion beats, and unclassified beats. The MIT-BIH was pre-split into training and test sets. In our implementation, the training set is further split into 80% training and 20% validation. Additionally, a reduced version of the MIT-BIH dataset was created by down-sampling the data to half the largest class size. This size reduction was done because of resource limitations and to allow for ample training time in the experiments. The PTB Diagnostic dataset provides two categories, distinguishing between normal or abnormal samples. Fig. 2 demonstrates the imbalance of classes for the training datasets.



(a) MIT-BIH Training Class Distribution (b) PTB diagnostic Training Class Distribution

Fig. 2: Class Distributions

C. Computational implementation

1) Data Preprocess

Leveraging on Pyspark's distributed parallelism, the CSV datasets were preprocessed to create training and testing Parquet files for faster access in our customized data preprocessed infrastructure. These files are loaded through a custom DataLoader to support training, validation and test of the various models. A custom Dataset and DataLoader extracts batches from the Parquet files and converts the data into tensors. The custom Dataset class inherits from the Pytorch's Dataset, ensuring seamless integration with PyTorch's DataLoader. The custom DataLoader supports capability to return both the raw data and the spectrogram transformed data, which is needed for the hybrid model. The custom DataLoader also splits the training data into training and validation sets, with random shuffling applied to each batch. Class weights were calculated for each label to account for the imbalanced dataset and applied to the CrossEntropyLoss function for the loss to adjust the penalty based on class distribution.

2) 1D CNN-Transformer

The architecture illustrated in Fig. 16 was implemented using Pytorch, incorporating 1D Convolution Neural Network layers, 1D Max Pooling and the Transformer Encoder. The positional embeddings described in [4] was implemented to preserve the sequence of the signals. The 1D CNN-Transformer model returns both the output score, represented by O_{TE}, and the feature vector, represented by H_{TE} in Fig. 16. The feature vector is the model's output of the Transformer Encoder layer, which will be needed for the Intermediate Fusion layer. Additionally, the Noam Optimization technique, as described in the referenced paper, was implemented as the optimization method for this model. CrossEntropyLoss with class weights was the loss function utilized during training. The model documented in the paper [2] was trained using a learning rate of 0.1, weight decay 0.0001, batch size of 16 and a dropout of 0.1 as shown in Table II.

3) 2D CNN

The spectrogram transformation was implemented using scipy and plotted as an image using matplotlib. As opposed to storing each spectrogram of the entire dataset in storage or memory, each sample is only transformed to a spectrogram image upon being loaded by the custom DataLoader. Therefore, there will only ever be a batch size number of images in memory at once, allowing the model to be more memory efficient and work with our large datasets.

All layers shown in Fig. 17 are implemented using pytorch. The 2D CNN model returns both the output score, represented by O_{TFR}, and the feature vector, represented by H_{TFR} in Fig. 17. The feature vector is the model's output of the last max pool layer, which will be needed for the Intermediate Fusion layer. The Adamax optimizer was implemented as the optimization method and the CrossEntropyLoss with class weights was the loss function utilized during training. The model documented in the paper [2] was trained using a learning rate of 0.001, weight decay 0.00001, batch size of 16 and a dropout of 0.2 shown in Table II.

4) Late Fusion

The Late Fusion model first loads the 2D CNN and 1D CNN-Transformer model and freezes them so they no longer get updated. Two weight vector parameters are initialized using pytorch's Parameter object. This object creates learnable attention weight vectors the size of the number of labels. These vectors get updated during back propagation to learn the best feature importance of the two model's output scores. During a forward pass, the inputs are passed into the two models to get both model's outputs. An element-wise multiplication on each of the outputs and their respective weights is performed and summed. This results in the final output score of the fused models.

5) Intermediate Fusion

Like the Late Fusion model, the Intermediate Fusion model loads and freezes the two individual models. During a forward pass, the inputs are passed into the two models and returns the model's feature vectors. Each feature vector is passed through a fully connected layer of size 64. Depending on the method chosen, the fully connected layers will either be concatenated, summed, or weighted summed. For the weighted sum, two learnable weights are initialized. These weights are trainable parameters, using pytorch's Parameter object. Once the combined feature representation method is performed, it is passed through a fully connected layer. The fully connected layer is of size 64 for the sum and weighted sum method, or of size 128 for the concatenation method. The output of the fully connected layer represents the final output score of the fused models.

Dataset	Model	Epochs	LR	WD	Batch size	DO	Opt.
PTB & MIT-BIH	1D CNN Trans-former	150	0.1	0.0001	16	0.1	Noam
PTB & MIT-BIH	2D CNN	30	0.001	0.00001	16	0.2	Adamax
MIT-BIH	Hybrid	10	0.0003	0.01	16	0.0	Adamax
PTB	Hybrid	10	0.001	0.01	16	0.0	Adamax

TABLE II: Hyper Parameters: LR (Learning rate), WD (Weight decay), DO (Dropout), Opt. (Optimizer) [2]

6) Configuration File Integration

In order to ensure the re-usability of code in the current customized infrastructure, a configuration file was created where parameters can be easily changed for the various models developed.

7) Loss Function

Cross Entropy loss with weights is used to consider the imbalance of the datasets.

8) Hardware

The models trained on the PTB Dataset were trained using an NVIDIA GeForce MX330 GPU with 4.0 GB of memory. The models trained on the MIT-BIH dataset were trained on an intel 13th gen i7-13700k CPU.

D. Code

The following is the GitHub link to the code: https://github.gatech.edu/egutierr3/Fall-2024-BD4H-project that was developed by us. For instructions to run the code, please reference the README.md in the root directory of the project.

1) Data Preprocess

- **download_data.py** Python script to download data from kaggle
- **data/dataloaders.py** Python and PySpark infrastructure for creating the customized dataset and dataloaders.
- **data/spark_utils.py** PySpark infrastructure to initialize a PySpark session, APIs for file processing with PySpark commands and APIs for creating parquet files.
- **data/data_transformation.py** Python APIs to transform the raw signal to extract the spectogram.
- **test/test_dataloaders.py** Python code to verify the data processing by creating batches.
- **test/test_data_transformation.py** Python code to verify the spectrogram transformation functionality.

2) 1D CNN-Transformer

- **models/one_d_cnn_transformer.py** Python and PyTorch code to create the 1D CNN-Transformer model.
- **test/test_one_d_cnn.py** Python code to verify the 1D CNN-Transformer model.

3) 2D CNN

- **models/CNN2D.py** Python and Pytorch to create the 2D CNN.
- **test/test_2d_cnn.py** Python code to verify the 2D CNN model.

4) Late Fusion

- **models/LateFusion.py** Python and Pytorch to create the late fusion hybrid model.
- **test/test_late_fusion.py** Python code to verify the late fusion hybrid model.

5) Intermediate Fusion

- **models/IntermediateFusion.py** Python and Pytorch to create the intermediate fusion hybrid model.
- **test/test_intermediate_fusion.py** Python code to verify the intermediate fusion hyrbid model.

6) Configuration File Integration

- **config/configuration.yaml** configuration file that contains variables for code. All test files use the parameters defined in this configuration file.
- **config/config_setup.py** Python script to process the configuration file to generate the variables.
- **test/test_config_setup.py** Python script to verify the use of configuration file.

7) Plots

- **plots/plots.py** Python file containing the various APIs to plot the metrics.

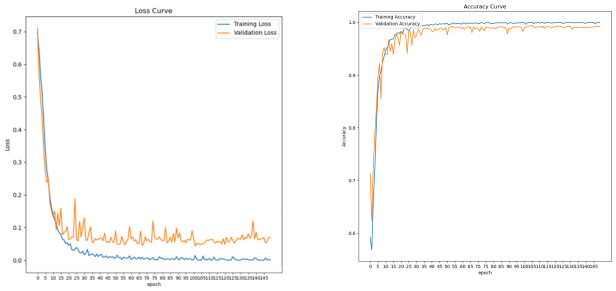
IV. Results

A. Training and Validation

The following subsections show the train and validation results for each of the four models previously discussed. Each of the following trained models will then be used with a test set to conduct the final results of Experiment 1 and Experiment 2.

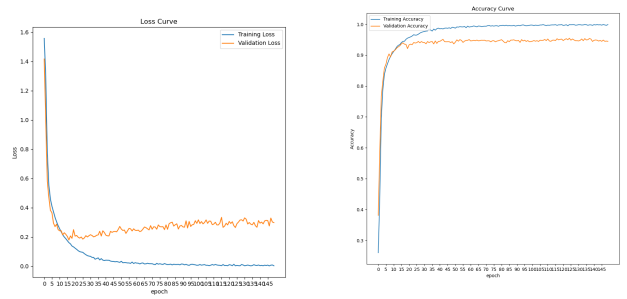
1) 1D CNN-Transformer

To verify the 1D CNN-Transformer, the model was trained and validated over 150 epochs on each dataset. The results in Fig. 3 show that the model accuracy plateaus after 50 epochs for the PTB dataset. Despite finding results similar to the original paper shown in Table IV, we found that 150 epochs was overkill and doesn't add much value to the model's learning capability. The MIT-BIH dataset results shown in Fig. 4 even show small indication of overfitting. The training curve begins to plateau after 75 epochs, but the validation curve reaches an optimal value after about 15 epochs on both curves. This result could be due to the reduced dataset discussed in the Data Description section.



(a) 1D CNN-Transformer Loss (b) 1D CNN-Transformer Accuracy

Fig. 3: 1D CNN-Transformer Results with PTB diagnostic dataset

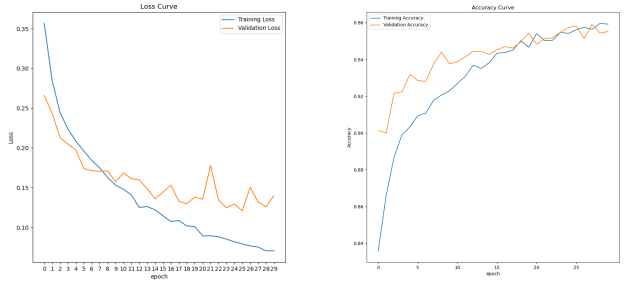


(a) 1D CNN-Transformer Loss (b) 1D CNN-Transformer Accuracy

Fig. 4: 1D CNN-Transformer Results with MITBIH diagnostic dataset

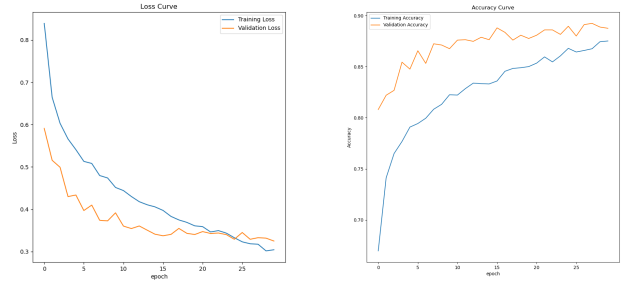
2) 2D CNN

The 2D CNN model was trained and validated over 30 epochs on each dataset. The results, shown in Fig. 5 and Fig. 6 both indicate no overfitting on either curves. Despite showing less overfitting, the 2D CNN slightly under performs the 1D CNN-Transformer model. This under performance aligns with the original paper's results.



(a) 2D CNN Loss (b) 2D CNN Accuracy

Fig. 5: 2D CNN Results with PTB diagnostic dataset

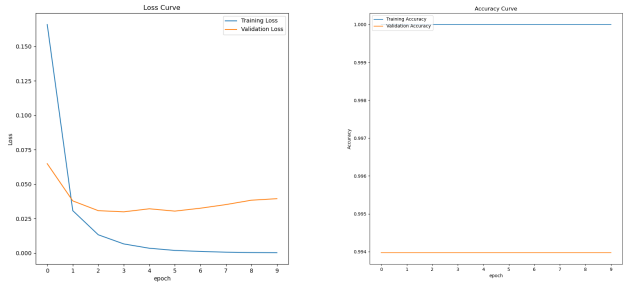


(a) 2D CNN Loss (b) 2D CNN Accuracy

Fig. 6: 2D CNN Results with MITBIH diagnostic dataset

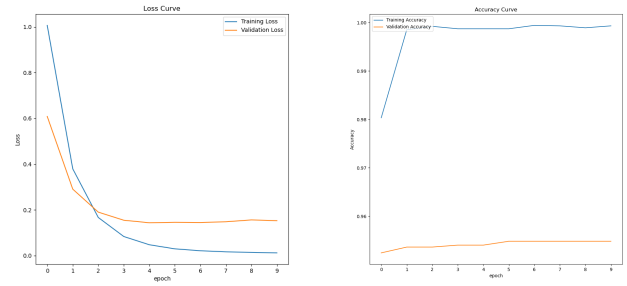
3) Late Fusion

The Late Fusion model was trained and validated over 10 epochs. The accuracy curves in Fig. 7 and Fig. 8 might appear to indicate overfitting, this perception is primarily due to the scale of the plots. In reality, the training and validation accuracy curves for both datasets are closely aligned. Furthermore, the seemingly limited learning progress is attributed to the small number of learnable parameters in these models. As discussed in the Late Fusion Model description, this architecture involves two vectors of learnable weights. Since these weights correspond to the number of labels, the model has very few parameters for the model to optimize.



(a) Late Fusion Loss (b) Late Fusion Accuracy

Fig. 7: Hybrid Late Fusion Results with PTB diagnostic dataset

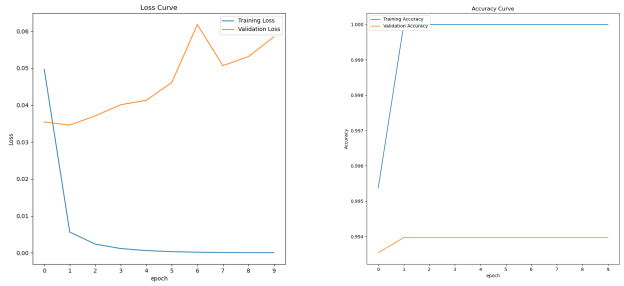


(a) Late Fusion Loss (b) Late Fusion Accuracy

Fig. 8: Hybrid Late Fusion Results with MITBIH diagnostic dataset

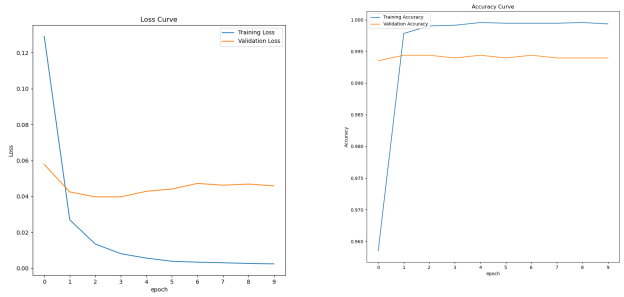
4) Intermediate Fusion

The Intermediate Fusion model was trained and validated over 10 epoch. Unlike the Late Fusion model, the Intermediate Fusion model has three fully connected layers with learnable parameters. Despite having three fully connected layers of parameters, some of the intermediate fusion model's learning curves plateau after a few epochs. This is likely due to the relatively small number of parameters in the three fully connected layers compared to the total number of parameters in the individual models. The reduced parameter count limits the model's capacity leading to faster convergence.



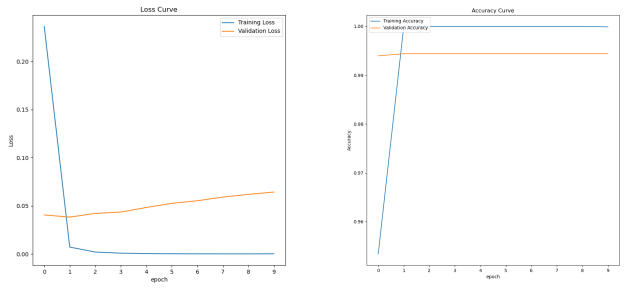
(a) Concatenation Intermediate Fusion Loss (b) Concatenation Intermediate Fusion Accuracy

Fig. 9: Hybrid Concatenation Intermediate Fusion Results with PTB diagnostic dataset



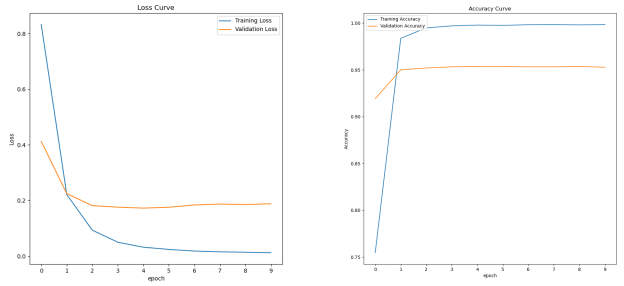
(a) Sum Intermediate Fusion Loss (b) Sum Intermediate Fusion Accuracy

Fig. 10: Hybrid Sum Intermediate Fusion Results with PTB diagnostic dataset



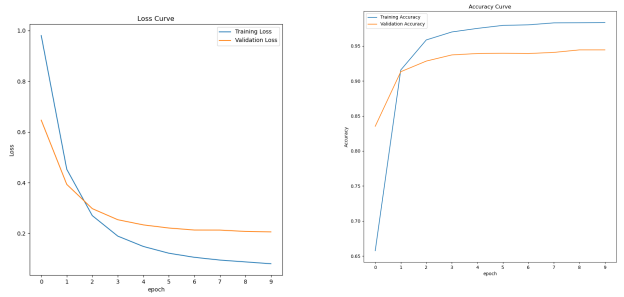
(a) Weighted Sum Intermediate Fusion Loss (b) Weighted Sum Intermediate Fusion Accuracy

Fig. 11: Hybrid Weighted Sum Intermediate Fusion Results with PTB diagnostic dataset



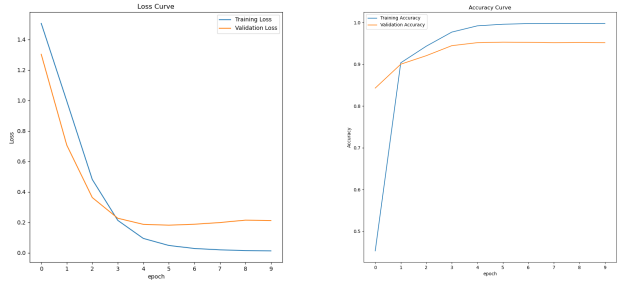
(a) Concatenation Intermediate Fusion Loss (b) Concatenation Intermediate Fusion Accuracy

Fig. 12: Hybrid Concatenation Intermediate Fusion Results with MITBIH diagnostic dataset



(a) Sum Intermediate Fusion Loss (b) Sum Intermediate Fusion Accuracy

Fig. 13: Hybrid Sum Intermediate Fusion Results with MITBIH diagnostic dataset



(a) Weighted Sum Intermediate Fusion Loss (b) Weighted Sum Intermediate Fusion Accuracy

Fig. 14: Hybrid Weighted Sum Intermediate Fusion Results with MITBIH diagnostic dataset

B. Experiment 1

The objective of the first experiment is to compare the performance of the 1D CNN-Transformer and 2D CNN models with the hybrid model that fuses the two models into a combined representation. The original paper's [2] results, shown in Table V, show that for all metrics and in both

datasets, the hybrid model outperforms the two individual models. Our reproduced results shown in Table IV demonstrate the exact same results. Although the metrics aren't exactly the same, our results for the PTB dataset are mostly within 1% margin of error, with the furthest being 4%. The MIT-BIH dataset on the other hand shows poorer performance compared to the original paper. This is largely due to the fact that we used a reduced subset of dataset because of hardware limitations. Despite this, the proportion of error between the models is the same between our results and the original paper's results.

Dataset	Model	Task	Time sec.	Hardware
PTB	1D CNN Transformer	Training & Validation	8110.32	GPU
	1D CNN Transformer	Testing	4.16	GPU
	2D CNN	Training & Validation	8150.89	GPU
	2D CNN	Testing	43.15	GPU
	Hybrid	Training & Validation	2068.46	GPU
	Hybrid	Testing	43.39	GPU
MIT-BIH-reduced	1D CNN Transformer	Training & Validation	2644.11	CPU
	1D CNN Transformer	Testing	19.34	CPU
	2D CNN	Training & Validation	914.43	CPU
	2D CNN	Testing	56.66	CPU
	Hybrid	Training & Validation	321.82	CPU
	Hybrid	Testing	58.41	CPU

TABLE III: Time Training for Experiment 1

Dataset	Model	MCC	F1 score	Accuracy
PTB	1D CNN Transformer	0.9848	0.9939	0.9939
	2D CNN	0.8912	0.9557	0.9554
	Hybrid Late Fusion	0.9865	0.9946	0.9946
MITBIH-Reduced	1D CNN Transformer	0.8120	0.9390	0.9300
	2D CNN	0.7363	0.9116	0.9012
	Hybrid Late Fusion	0.8412	0.9494	0.9435

TABLE IV: Test Results Experiment 1

C. Experiment 2

The objective of the second experiment is to highlight the advantages of late fusion with learnable weights compared to other fusion methods. The original paper's [2] results, shown in Table VIII, show that for all metrics and in both datasets, the late fusion model outperforms all the intermediate fusion models. Our reproduced results shown in Table VII demonstrate the exact same results. It's also interesting to

Dataset	Model	MCC	F1 score	Accuracy
PTB	1D CNN Transformer	97.92±0.28	98.96±0.14	99.16±0.11
	2D CNN	93.42±2.27	96.66±1.20	97.32±0.91
	Hybrid	99.29±0.21	99.65±0.10	99.71±0.08
MIT-BIH	1D CNN Transformer	93.17±0.70	89.44±0.99	97.87±0.24
	2D CNN	91.26±0.76	86.94±1.39	97.34±0.26
	Hybrid	94.63±0.29	91.28±0.54	98.37±0.09

TABLE V: Original Results Experiment 1 [2]

note that our intermediate fusion models on the PTB dataset outperform those from the original paper. Even though our experiment came to the same conclusions as the original paper's experiment, the late fusion model only did marginally better than the intermediate fusion models. The improved intermediate fusion performance could indicate that our 1D CNN-Transformer and 2D CNN has better feature representation compared to the original paper. Similar to experiment 1, the MIT-BIH dataset underperformed the model results from the original paper because we used a reduced subset of the dataset.

Dataset	Fusion Type	Task	Time sec.	Hardware
PTB	Concat	Training & Validation	2301.99	GPU
	Concat	Testing	73.21	GPU
	Sum	Training & Validation	2349.38	GPU
	Sum	Testing	43.27	GPU
	Weighted-Sum	Training & Validation	2221.73	GPU
	Weighted-Sum	Testing	54.23	GPU
	Hybrid	Training & Validation	2068.46	GPU
	Hybrid	Testing	43.39	GPU
MIT-BIH-reduced	Concat	Training & Validation	320.23	CPU
	Concat	Testing	59.24	CPU
	Sum	Training & Validation	318.50	CPU
	Sum	Testing	59.68	CPU
	Weighted-Sum	Training & Validation	322.32	CPU
	Weighted-Sum	Testing	58.89	CPU
	Hybrid	Training & Validation	321.82	CPU
	Hybrid	Testing	58.41	CPU

TABLE VI: Time Training for Experiment 2

V. Discussion

A. Conclusions

Based on our results of both experiments, we successfully reproduced comparable results and reached the same conclusion as the original paper where the Hybrid Late Fusion model outperforms the other models.

Dataset	Fusion Type	MCC	F1 score	Accuracy
PTB	Concat	0.9848	0.9939	0.9939
	Sum	0.9822	0.9929	0.9929
	Weighted-Sum	0.9848	0.9939	0.9939
	Hybrid	0.9865	0.9946	0.9946
MITBIH-Reduced	Concat	0.8173	0.9414	0.9323
	Sum	0.7967	0.9335	0.9233
	Weighted-Sum	0.8074	0.9375	0.9278
	Hybrid	0.8412	0.9494	0.9494

TABLE VII: Test Results Experiment 2

Dataset	Fusion Type	MCC	F1 score	Accuracy
PTB	Concat	92.91±2.61	96.42±1.33	97.11±1.05
	Sum	92.12±2.33	96.02±1.19	96.78±0.99
	Weighted-Sum	92.74±2.01	96.35±1.00	97.06±0.81
	Hybrid	99.29±0.21	99.65±0.10	99.71±0.08
MIT-BIH	Concat	91.51±0.79	86.93±1.10	97.42±0.27
	Sum	91.89±0.47	87.50±0.87	97.55±0.15
	Weighted-Sum	91.56±0.72	86.70±1.13	97.44±0.24
	Hybrid	94.63±0.29	91.28±0.54	98.37±0.09

TABLE VIII: Original Results Experiment 2 [2]

B. Challenges

One of the biggest challenges with reproducing these results was the size of the dataset with limited hardware and memory resources. As previously discussed, we were limited to only a reduced subset of the MIT-BIH dataset. This was primarily because of the 150 epoch parameter for the 1D CNN-Transformer. It was estimated that running all 150 epochs with the entire MIT-BIH dataset would take 20 hours of 100% hardware. However, despite seeing lower performance on the reduced dataset, we still were able to draw the same conclusions from the experiments.

C. Suggestions

One suggestion for the authors of the original paper would be to unfreeze the parameters of the 1D CNN-Transformer and 2D CNN models during the fusion models. This approach could evaluate whether allowing the full models to learn while being trained together impacts the performance. Additionally, the paper does not specify whether hyperparameter tuning was employed in their evaluations. In our results, the accuracy and loss graphs for the models plateau after a certain number of epochs, suggesting that using hyperparameter tuning tools, such as Ray Tune or Optuna, could help identify optimal values for epochs, learning rate, and other parameters to achieve improved results. We also recommend using a GPU with sufficient memory to determine an optimal batch size.

In the original paper, thirty-two filters were applied to the spectrogram in the convolutional layers of the 2D CNN. However, in our implementation, we used only eight filters, which resulted in fewer trainable parameters and reduced memory usage.

VI. Final Presentation

The following is the link for the final presentation of the overall project https://www.youtube.com/watch?v=ktww2O-o0WA.

REFERENCES

[1] Manuel Martínez-Sellés. Current and future use of artificial intelligence in electrocardiography. *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, 2018. PDF available at http://dx.doi.org/10.3390/jcdd10040175. 1

[2] Yamil Vindas, Blaise Kevin Guepie, Marilys Almar, Emmanuel Roux, and Philippe Delachartre. An hybrid cnn-transformer model based on multi-feature extraction and attention fusion mechanism for cerebral emboli classification. *Proceedings of the 7th Machine Learning for Healthcare Conference*, 182:270–296, 2022. PDF available at https://proceedings.mlr.press/v182/vindas22a/vindas22a.pdf. 1, 2, 3, 6, 7, 8, 10, 11

[3] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. Ecg heartbeat classification: A deep transferable representation. *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, 2018. Data available at https://www.kaggle.com/datasets/shayanfazeli/heartbeat. 1, 2

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, 2017. PDF available at http://arxiv.org/abs/1706.03762. 1, 3

[5] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. Allennlp: A deep semantic natural language processing platform. In *AllenNLP: A Deep Semantic Natural Language Processing Platform*, 2017. https://docs.allennlp.org/main/api/training/learning_rate_schedulers/noam/. 1

APPENDIX A
HYBRID MODEL PIPELINE

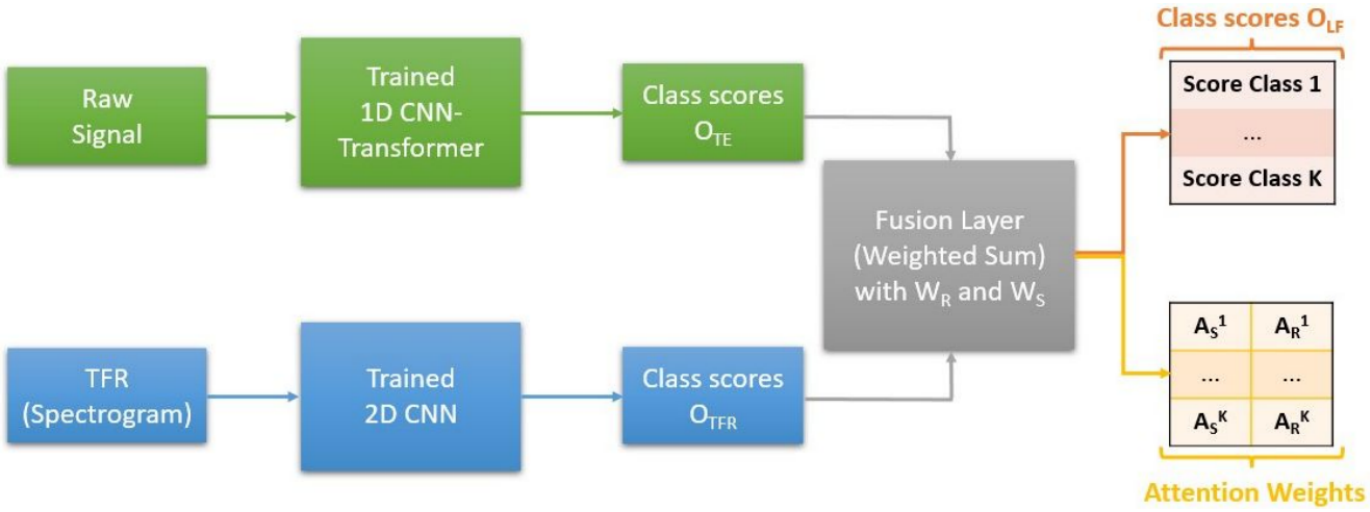


Fig. 15: Hybrid model Pipeline [2]

APPENDIX B
1D CNN-TRANSFORMER ARCHITECTURE

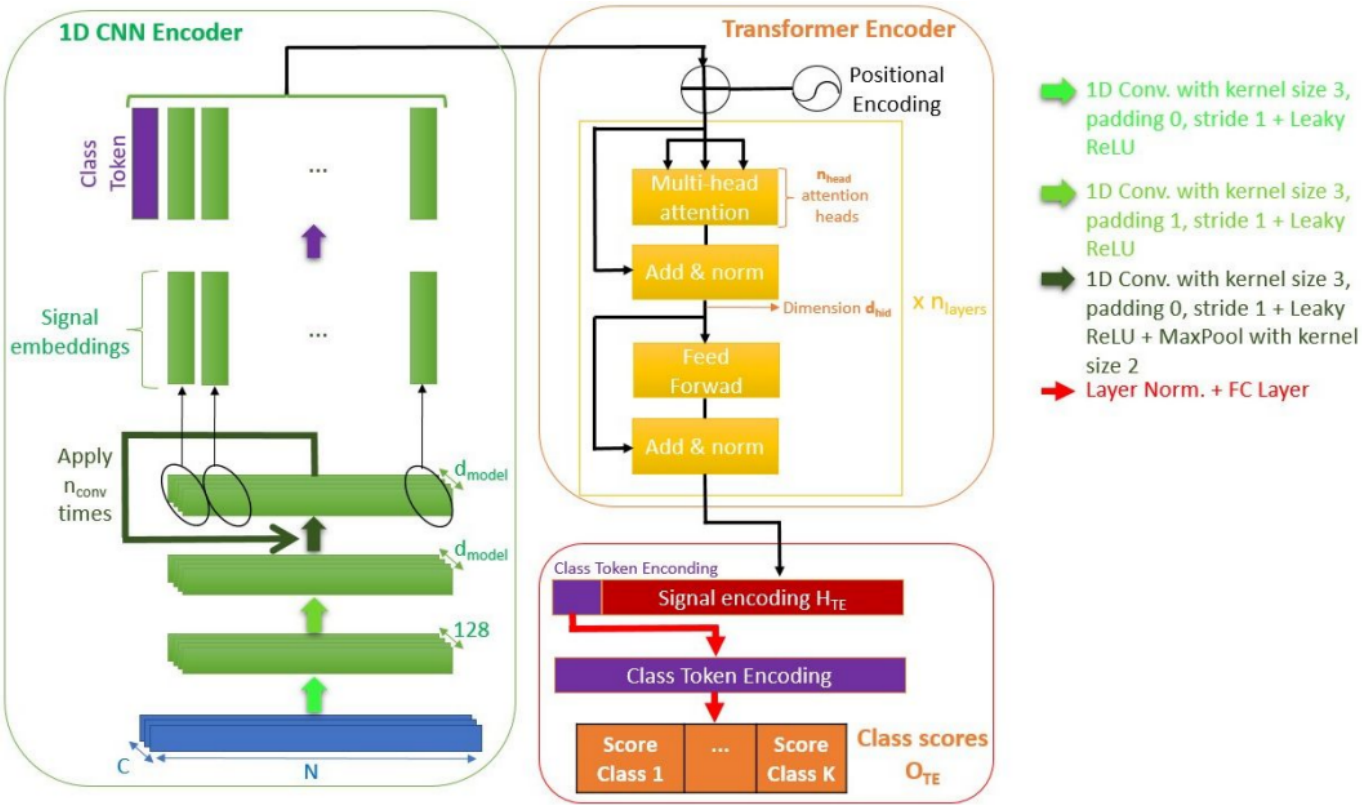


Fig. 16: Hybrid 1D CNN-Transformer architecture [2]

APPENDIX C
2D CNN ARCHITECTURE

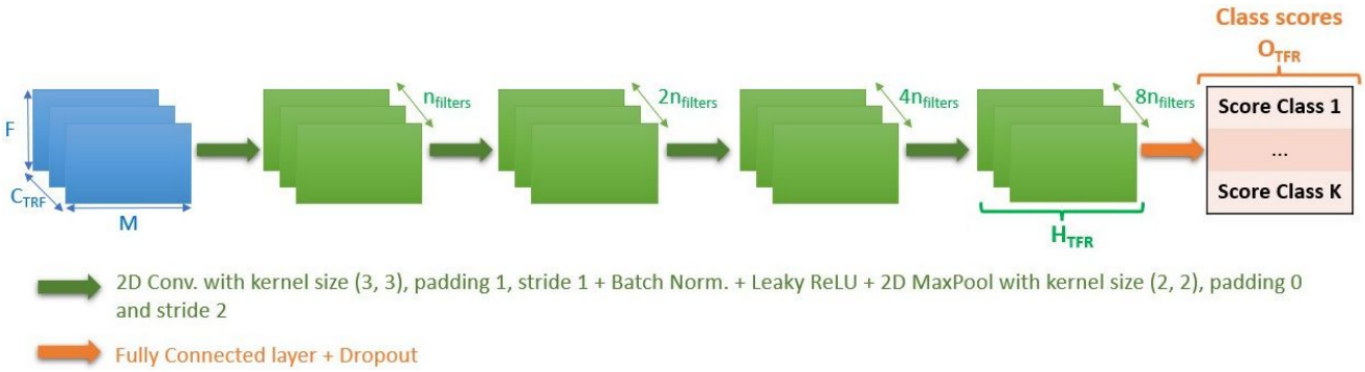


Fig. 17: 2D CNN architecture [2]