

# Machine Learning Engineer Nanodegree

## Capstone Project

---

Alexander Villasoto

April 12, 2020

## I. Definition

### Project Overview

One of the sectors that deep learning has made a tremendous impact is in healthcare<sup>1</sup>. In fact, researches on the medical sector involve deep learning methods in their analyses. Anyone could agree that it is imperative that disease diagnosis early on the treatment is equally if not more important as the treatment itself. Therefore, these advancements when applied to the health sector are really helpful in saving people's lives since one of the benefits of a carefully trained deep learning model is its accuracy and speed in generating results. Truly, this will not only benefit patients but doctors as well<sup>2</sup>.

Out of all the diseases that need utmost priority of the medical sector as well as researchers in the field of healthcare, heart disease is the one that needs careful attention. In the United States, heart disease is the leading cause of death for men, women and primary racial and ethnic groups<sup>3</sup>. Alarming, one person dies every 37 seconds in the United States from heart disease and about 647,000 Americans die from heart disease each year<sup>3,4</sup>. Fortunately, milestones of deep learning and artificial intelligence have lead several researchers to create fast and accurate models for heart disease detection<sup>1</sup>.

One of the ways to do this is through the use of previously-available digitized electrocardiogram (ECG/EKG) data and apply deep learning methods to classify specific heart disease<sup>5</sup>. With the already available dataset and publicly available paper tackling heart disease detection, the author decided to implement a deep learning model that would attain the same level of reported accuracy if not more using a state-of-the-art deep learning framework.

---

<sup>1</sup> Ahuja A. S. (2019). The impact of artificial intelligence in medicine on the future role of the physician. PeerJ, 7, e7702. <https://doi.org/10.7717/peerj.7702>

<sup>2</sup> Aljanabi, Maryam & Qutqut, Mahmoud & Hijawi, Mohammad. (2018). Machine Learning Classification Techniques for Heart Disease Prediction: A Review. International Journal of Engineering and Technology. 7. 5373-5379. 10.14419/ijet.v7i4.28646.

<sup>3</sup> Heron, M. Deaths: [Leading causes for 2017 \[PDF – 3 M\]](#). National Vital Statistics Reports;68(6). Accessed November 19, 2019.

<sup>4</sup> Fryar CD, Chen T-C, Li X. [Prevalence of uncontrolled risk factors for cardiovascular disease: United States, 1999–2010 \[PDF-494K\]](#). NCHS data brief, no. 103. Hyattsville, MD: National Center for Health Statistics; 2012. Accessed May 9, 2019.

<sup>5</sup> Kachuee, Mohammad, Shayan Fazeli, and Majid Sarrafzadeh. "ECG Heartbeat Classification: A Deep Transferable Representation." 2018 IEEE International Conference on Healthcare Informatics (ICHI) (2018): n. pag. Crossref. Web.

## Problem Statement

The problem revolves around the utilization of a publicly available digitized EKG dataset to create a deep learning model that would attain the same level of the paper's reported accuracy, 93.4% on the MIT-BIH dataset if not more using the PyTorch framework <sup>5</sup>. The author tried and replicated the model definition of the paper with several improvements to prevent overfitting and employ functions for training normalization. The author implemented implement train -> test -> validation strategies and reported the results on a per class basis to prevent overfitting <sup>6</sup>. Since the problem is multiclass classification, accuracy and loss are the major metrics that the author needed to be increased and decreased respectively. Ultimately, the author made sure that the results are communicated via reproducible document.

## Metrics

Since the project is primarily a multiclass classification problem, the metric that we are trying to minimize is the negative log likelihood loss (NLLLoss) <sup>7</sup>. In this metric, we take the sum of log loss

$$-\sum_{c=1}^M y_{o,c} \log(p_{o,c})$$

values of each class prediction in the observation where  $\Sigma$  is shorthand for summation or in our case the sum of all log loss values across classes and  $c = 1$  is the first class of the summation as starting point <sup>8</sup>. We take the negative log in this case to achieve easy metric for comparison since the positive log of numbers that are less than one returns negative values <sup>8,9</sup>. All of these can be conveniently achieved using the criterion function available in PyTorch by comparing the batch predictions and the ground truth labels per iteration given that the output of the model are softmax values <sup>7</sup>.

$$Accuracy_{class} = \frac{\text{number of correct predictions per class}}{\text{total observations per class}}$$

Additionally, we evaluate the performance of the model with the least test/validation loss from the aforesaid metric via accuracy. Accuracy is just the sum of the correctly-identified classes divided by the total observations. For this, we compute the overall accuracy as well as the accuracy per class. These are all computed in the ***evaluate\_model*** function definition (see the Model Evaluation and Validation section below). The inspiration came from confusion\_matrix and classification\_report functions from sklearn.metrics package <sup>10</sup>.

---

<sup>6</sup> "Subject: What are the population, sample, training set, design set, validation set, and test set?", Neural Network FAQ, part 1 of 7: Introduction ([txt](#)), comp.ai.neural-nets, Sarle, W.S., ed. (1997, last modified 2002-05-17)

<sup>7</sup> "Torch.nn¶." Torch.nn - PyTorch Master Documentation, [pytorch.org/docs/stable/nn.html#nllloss](https://pytorch.org/docs/stable/nn.html#nllloss).

<sup>8</sup> Log Loss. [wiki.fast.ai/index.php/Log\\_Loss](https://wiki.fast.ai/index.php/Log_Loss).

<sup>9</sup> Tovar, Alvaro Durán. Negative Log Likelihood. 4 Mar. 2020, [medium.com/deeplearningmadeeasy/negative-log-likelihood-6bd79b55d8b6](https://medium.com/deeplearningmadeeasy/negative-log-likelihood-6bd79b55d8b6).

<sup>10</sup> "API Reference¶." Scikit, [scikit-learn.org/stable/modules/classes.html](https://scikit-learn.org/stable/modules/classes.html).

## II. Analysis

### Data Exploration

The author's project is all about creating a model that categorizes and predicts several heartbeat observations. He will use the MIT-BIH Arrhythmia Dataset, which is readily available on Kaggle <sup>11</sup>. For the MIT-BIH Arrhythmia Dataset, all the samples are cropped, downsampled and padded with zeroes when necessary to the fixed dimension of 188 as per the Kaggle dataset remark which can be found on the dataset above. The dataset that is provided is divided into separate CSV files, one for training and one for testing. The author is responsible for creating training, testing and validation datasets throughout the modeling process.

The dataset has five classes, 'N' assigned to 0, 'S' assigned to 1, 'V' assigned to 2, 'F' assigned to 3 and 'Q' assigned to 4. Specifically, classes above refer to the beat annotations enumerated below <sup>12</sup>:

1. N - Normal beat
2. S - Supraventricular premature or ectopic beat (atrial or nodal)
3. V - Premature ventricular contraction
4. F - Fusion of ventricular and normal beat
5. Q - Unclassifiable beat

Additionally, each observation in this particular dataset are actual recorded heartbeats of 47 different subjects at a sampling rate of 360Hz annotated by at least two cardiologists under five classes mentioned above in accordance with the Association for the Advancement of Medical Instrumentation (AAMI) standard <sup>13</sup>. The last column indicates how each heartbeat is classified, that is the classification of a specific beat per the beat annotations above <sup>11</sup>.

### Exploratory Visualization

Before starting with the project, the author made sure that the dataset is fit for the modeling stage. Visualizing the distribution of the dataset through pie chart, it is evident that the dataset is highly unbalanced (see figure 1). Specifically, about 90000 observations cover the normal beat class, followed by unclassified beat with a little more than 8000.

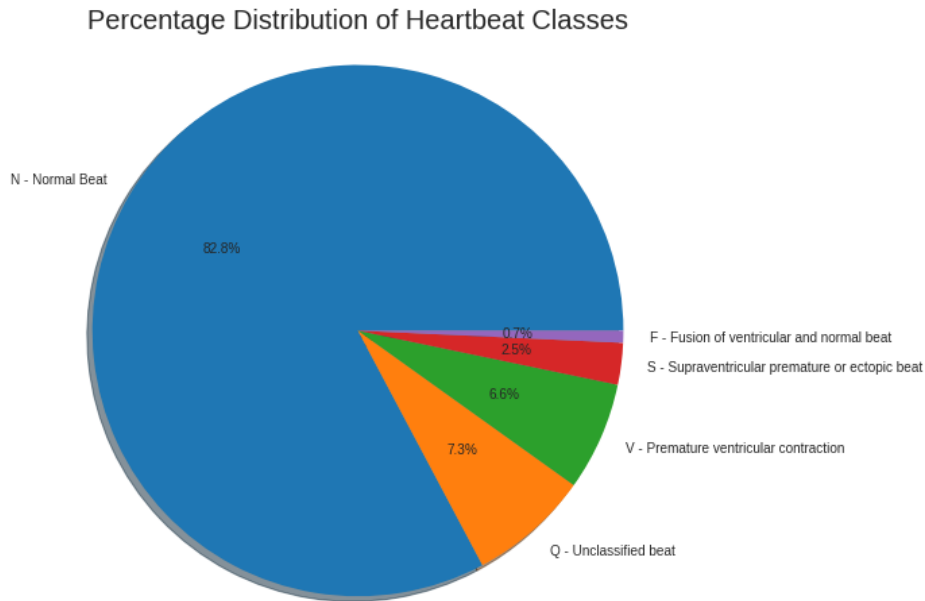
---

<sup>11</sup> Fazeli, S. (2018, June). ECG Heartbeat Categorization Dataset. Retrieved April 7, 2020 from <http://kaggle.com/shayanfazeli/heartbeat>.

<sup>12</sup> PhysioBank Annotations, [archive.physionet.org/physiobank/annotations.shtml](http://archive.physionet.org/physiobank/annotations.shtml).

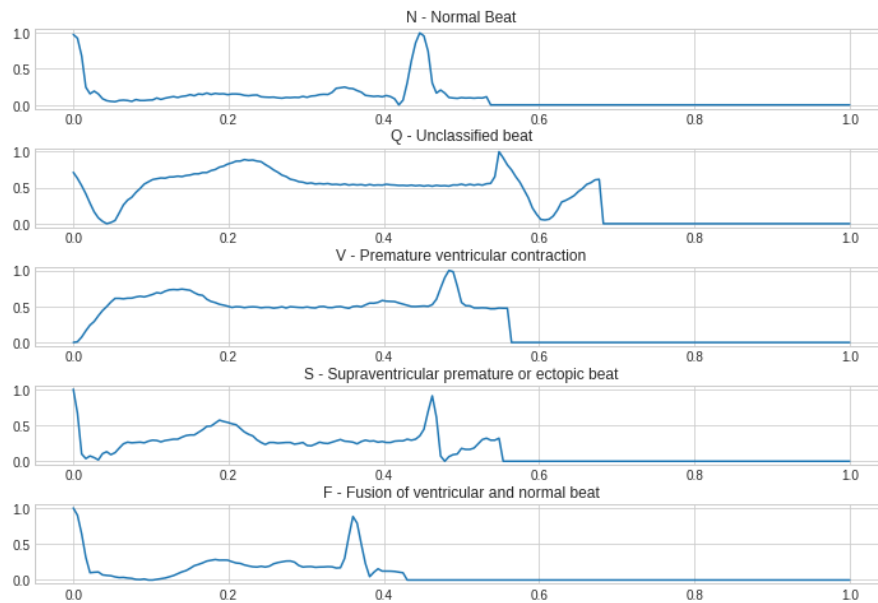
<sup>13</sup> Luz, Eduardo José da S., et al. "ECG-based heartbeat classification for arrhythmia detection: A survey." *Computer methods and programs in biomedicine* 127 (2016): 144-164.

Premature ventricular contraction with 6.6% has 7200 observations, supraventricular premature or ectopic beat with 2779 observations and lastly, fusion of ventricular and normal beat with only 803 observations.



*Fig. 1: The chart shows the distribution of observations per class,  $N$  being the dominant class covering 82.8% of the total observations.*

Note that these came from both train and test CSVs of the public dataset since evidently, we have limited number of observations per class particularly for fusion of ventricular and normal beat. Even so, 800 observations are still small for deep learning tasks so the author needs to upsample each class.



*Fig. 2: Visualizing heartbeat per class*

This may overfit class F (fusion of ventricular and normal beat b) but its influence to the model will be on par with other classes.

After we separate actual observation and their labels, we will visualize each of the beat classes mentioned above using line graphs (see fig, 2). Since it is clear that every column in the dataset represent beat values and the dataset has already been prepared with padded zero values, no further amount of data preparation or imputation is required.

## Algorithms and Techniques

The author chose to implement a deep learning algorithm instead of using classical machine learning algorithms in this project. For one thing, the paper which is the basis for this project which generously discussed their own model implementation is a deep learning one, specifically a custom convolutional neural network. Another reason is the nuances of the signals that need to be distinguished by the model are too complicated for classical machine learning algorithms to find separations for. Finally, it is also an opportunity for the author to test his knowledge of building deep learning models that can effectively be applied to a real-world problem.

In the modeling process, the author defines the loss function/error criterion appropriate for this specific problem and the model optimizer. Given the augmented dataset with dataset size appropriate for CNN task, the author creates data loaders for both training and validation dataset with a carefully-selected batch size. After these, he defines model definitions, one derived from the paper and one is the custom one with input features and output dimension as their parameters. A fair number of epochs is selected and training loops for both models are defined.

Inside each training epoch, model is initialized for training, gradients are zeroed out, model is fitted with batch data spitted out from the dataloader iterator, loss deviation is computed using a previously-selected loss function, backward pass is called and weights are updated via previously-selected model optimizer. Ultimately, training loss per batch is computed and are accumulated until batches of data are exhausted in that epecific iteration.

Sanity check is done per epoch via model evaluation. The author does this by putting the model in the evaluation mode. In this case, outstanding model weights are frozen for faster inference. We do the same workflow as the training per batch by getting the output from the model, computing the loss and accumulating the validation loss per batch. At the very end of each training epoch, training and validation loss are averaged and reported.

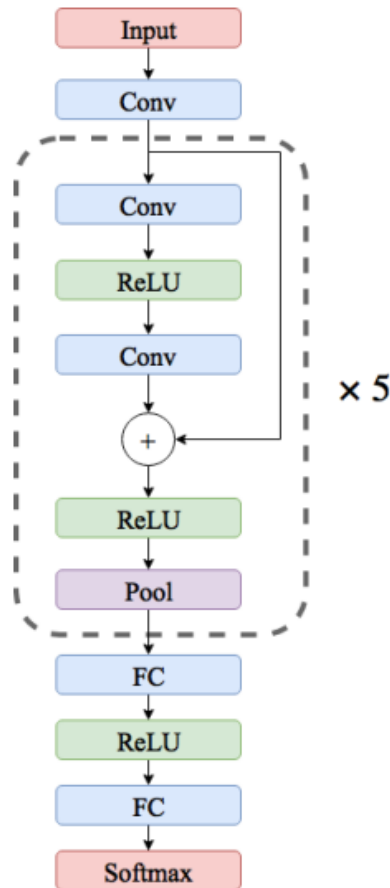
All of these processes are done using convenience helper methods from PyTorch framework from data loading, selecting metrics and hyperparameters and the actual training the model. Since PyTorch has native support for CUDA, the author uses GPU acceleration for faster training. Actual training has been done in the Kaggle workspace which provide free GPU access. Aside from these the author has the convenience to access the dataset without the unnecessary migration of data into different environments<sup>14</sup>.

---

<sup>14</sup> Dansbecker. "Running Kaggle Kernels with a GPU." Kaggle, Kaggle, 11 May 2018, [www.kaggle.com/dansbecker/running-kaggle-kernels-with-a-gpu](https://www.kaggle.com/dansbecker/running-kaggle-kernels-with-a-gpu).

## Benchmark Model

The said paper used several one-dimensional convolutional layers with RELU activations on the learning layer and linear fully-connected layer at the end (see the figure below). The paper reported 93.4% accuracy across all test observations for the MIT-BIH dataset presented above. This is what the author is trying to achieve. An accuracy on par if not more than the reported accuracy of the paper.



*Fig. 3: Architecture of the benchmark network from the paper*

Also, the model had softmax activation at the very end to enable negative log likelihood loss for the loss function and adaptive learning rate (ADAM) for model optimizer with learning rate of 0.001. To test whether the model is improving, the author used validation loss as metric. The author also assessed the model performance by recording the training and validation loss across all observations and saved the best model every time the validation loss decreases. The author trained both the models for 50 epochs with batch size of 32. The choice for the loss function, optimization scheme and metric are pretty much arbitrary and are not discussed on the paper but are fairly common to be used in multiclass classification tasks for deep learning<sup>15</sup>. Finally, he made the final check of accuracy across all classes and observations in the test set using the best model saved during the training process.

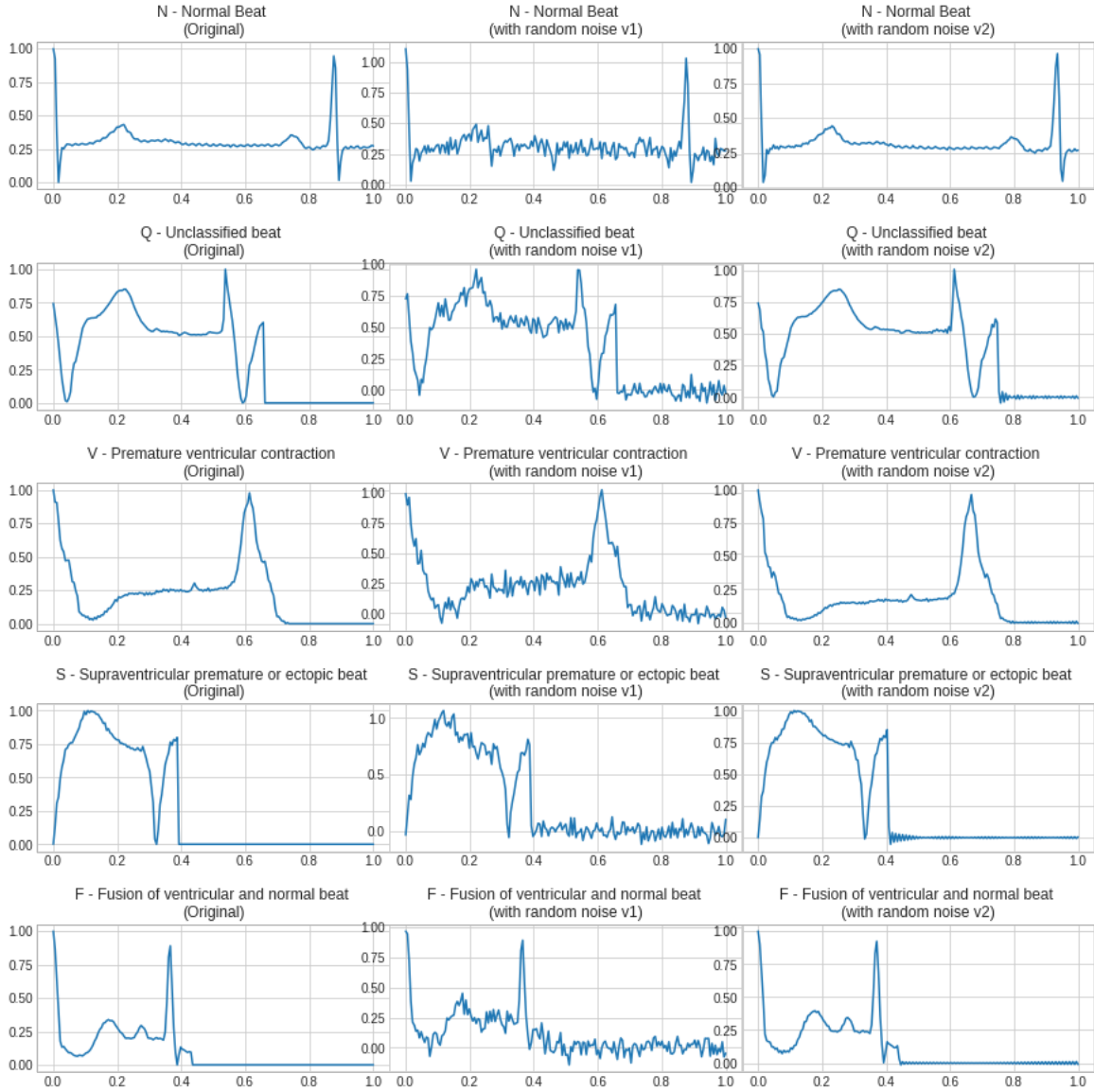
---

<sup>15</sup> “Deep Learning with PyTorch¶.” Deep Learning with PyTorch - PyTorch Tutorials 1.4.0 Documentation, [pytorch.org/tutorials/beginner/nlp/deep\\_learning\\_tutorial.html](https://pytorch.org/tutorials/beginner/nlp/deep_learning_tutorial.html).

# III. Methodology

## Data Preprocessing

Given the dataset specification, the author made sure that the dataset, given its predictors (187 values per signal) is fit for the modeling stage. As said before, the author needed to upsample each classes since the dataset is inherently unbalanced. Thus, he resampled all the classes to 10000 observations each.



*Fig. 4: Comparison of Resulting Feature Creation and Upsampling Strategies Employed in the Dataset*

Since resampling 10000 observations would likely overfit classes with scarce observations in them, the author employed Gaussian noise to each beat observation by an acceptable amount of deviations, in this case 0.05 standard deviations above and below the mean. Another way is to add random signal amplification and stretching which does not distort each signal but ever so slightly shift or amplify each data point via calculated randomized chances.

He will also make sure that each of the categories mentioned are equally distributed, implementing feature creation, feature engineering and resampling strategies when necessary as response to the prior data exploration.

Either way, the author will record the performance of the model that uses either of the strategies mentioned. After the aforesaid strategies, the author will then divide the dataset into two parts, training set which is 80% of the total observations and 20% for the validation and testing.

## Implementation

### Model Building

Based on the model definition described on the paper, the author built two models, testing both of the resampling strategies mentioned above. One is the direct copy of the model definition described in the paper and the other is an attempt to improve model performance by adding improvements like extra layers to achieve greater accuracy, decrease overfitting and employ functions for training normalization. We use PyTorch framework in both cases.

Layer (type)	Output Shape	Param #
Conv1d-1	[-1, 32, 183]	352
Conv1d-2	[-1, 32, 183]	352
Conv1d-3	[-1, 32, 183]	5,152
MaxPool1d-4	[-1, 32, 90]	0
Conv1d-5	[-1, 32, 90]	5,152
Conv1d-6	[-1, 32, 90]	5,152
MaxPool1d-7	[-1, 32, 43]	0
Conv1d-8	[-1, 32, 43]	5,152
Conv1d-9	[-1, 32, 43]	5,152
MaxPool1d-10	[-1, 32, 20]	0
Conv1d-11	[-1, 32, 20]	5,152
Conv1d-12	[-1, 32, 20]	5,152
MaxPool1d-13	[-1, 32, 8]	0
Linear-14	[-1, 32]	8,224
Linear-15	[-1, 32]	1,056
Linear-16	[-1, 5]	165
LogSoftmax-17	[-1, 5]	0
Total params: 46,213		
Trainable params: 46,213		
Non-trainable params: 0		
Input size (MB): 0.02		
Forward/backward pass size (MB): 0.25		
Params size (MB): 0.18		
Estimated Total Size (MB): 0.45		

*Fig. 5: Training Parameters of the First Model Based on the Paper*



For the former model, we see group of convolutions and max pooling layers the same on the paper (see fig. 3). These group of convolutions are coupled with ReLU activations and max pooling layers, taking into account the addition operator in between that acts as “identity shortcut connection”, intentionally skipping a number of convolution layers to prevent the vanishing gradient problem <sup>16</sup>.

Layer (type)	Output Shape	Param #
Conv1d-1	[-1, 128, 183]	1,408
Conv1d-2	[-1, 128, 183]	1,408
Conv1d-3	[-1, 128, 183]	82,048
MaxPool1d-4	[-1, 128, 90]	0
Conv1d-5	[-1, 128, 90]	82,048
Conv1d-6	[-1, 128, 90]	82,048
MaxPool1d-7	[-1, 128, 43]	0
Conv1d-8	[-1, 128, 43]	82,048
Conv1d-9	[-1, 128, 43]	82,048
MaxPool1d-10	[-1, 128, 20]	0
Conv1d-11	[-1, 128, 20]	82,048
Conv1d-12	[-1, 128, 20]	82,048
MaxPool1d-13	[-1, 128, 8]	0
Conv1d-14	[-1, 128, 8]	82,048
Conv1d-15	[-1, 128, 8]	82,048
MaxPool1d-16	[-1, 128, 2]	0
Linear-17	[-1, 32]	8,224
Linear-18	[-1, 5]	165
LogSoftmax-19	[-1, 5]	0
Total params: 749,637		
Trainable params: 749,637		
Non-trainable params: 0		
Input size (MB): 0.02		
Forward/backward pass size (MB): 1.01		
Params size (MB): 2.86		
Estimated Total Size (MB): 3.89		

*Fig. 6: Training Parameters of the Second Model with Deeper Layers*

For the latter model, the author added another group of layers with identity shortcut connection and max pooling layers. The thinking is since stacking layers inherently preserves gradients through layer stacking, deeper models like this should not degrade the network performance. And since deeper networks tend to find more patterns than the shallower ones <sup>17</sup>, the author argues that this model would provide better accuracy. Also, as you can see, the author chose to create model with more parameters given by the increase output convolution channels (second dimension per layer, from 32 to 128) to ensure that more patterns can be found per class.

<sup>16</sup> K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015).

<sup>17</sup> Mhaskar, H., Liao, Q., Poggio, T.: Learning functions: when is deep better than shallow. arXiv:1603.00988 (2016)

## Model Training

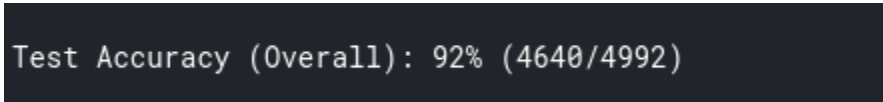
For the model training, the author trained both models with strategies presented on the Algorithms and Techniques section above. In every decrease of the validation loss, the model that lead to that decrease will be saved which will then be used for model evaluation. The author recorded the average loss per batch and show the progression of the training process every validation batches. In this phase, the author used the splitted training set that was generated during the data wrangling stage (discussed on Data Preprocessing section above).

Since the author had to report four modeling results due to the combination of both data preprocessing and the two models, he defined a function called ***train\_by\_model\_and\_custom\_loader*** that accepts a PyTorch model, train and validation loaders from PyTorch DataLoader class, loss function criterion, optimizer, string literal for the best model name, number of epochs and a boolean value which determine if the device used is capable of training on GPU.

Inside the aforesaid function, model is converted to float and shifted to CUDA given that the *train\_on\_gpu* parameter is true. In each training epoch, the converted and shifted model is initialized for training, gradients are zeroed out, model is fitted with batch data from the train loader, loss per batch is computed using a supplied loss function, backward pass is called and weights are updated via the supplied model optimizer (more information can be found in the Algorithms and Techniques section above). After each training epoch has been completed, the function returns the recorded validation losses (see the Metrics section) to compare epoch losses for all the trained models via line graphs.

## Refinement

Initially, the author targeted to only have one model for the project. The suggested network architecture of the paper had received satisfactory results with a test accuracy of 92 percent but not the target 93.4% from the paper (see figure below). Also, Gaussian Noise was the only upsampling method that was adopted during the process and test set was not augmented to the whole train -> test -> valid pipeline which contributed to the scarcity of the dataset samples, not ideal for deep learning tasks like this <sup>18</sup>. Also, training has only been done on 10 epochs with learning rate of 0.01 in which, given that the training still has a trend to decrease could be increased more by both changing the number of epochs and decreasing the learning rate for the previously-selected optimizer.



Test Accuracy (Overall): 92% (4640/4992)

*Fig. 7: Initial Accuracy Got from the Early Model Before Refinement*

With all these observations i.e. limited number of layers, low number of output channels, lack of alternative feature creation strategies and low number of training epochs lead the author to create a new model as well as derive alternative feature creation and upsampling strategy. For comparison, the author retained the original model and feature creation strategy as benchmark to objectively quantify how well the new model improved.

---

<sup>18</sup> Feng, Shuo, Huiyu Zhou, and Hongbiao Dong. "Using deep neural network with small dataset to predict material defects." *Materials & Design* 162 (2019): 300-310.

## IV. Results

### Model Evaluation and Validation

In model evaluation, the testing dataset was used to finally test the good model between two model definitions. The author defined a function called *evaluate\_model* that accepts the model that was trained, test loader, and loss function as before and the best model name that was supplied in *train\_by\_model\_and\_custom\_loader* used in the training process.

Inside the function, model is loaded with the state dictionary of the loaded PyTorch model using the best model name. Model is explicitly declared to be in evaluation mode where the author froze the weights in the already loaded model which had already been trained during the training process. The usual prediction and loss computation is employed in every batch of data yielded by the test loader and the accuracy per batch is accumulated. After the evaluation mode, the overall test loss is printed out, and accuracy per class are printed as well including the overall accuracy (see Metrics section above).

To review, the author made four models, specifically *model\_1* with the original paper's neural network architecture with Gaussian Noise upsampling, *model\_2* with the same model but with amplification and stretching strategy for upsampling, *model\_3* that uses the modified version of a neural network architecture with Gaussian Noise upsampling and finally, *model\_4* with the same model as model 3 with the amplification and stretching strategy for upsampling. The result of these trained models are as follows:

```
Test Loss: 0.128242

Test Accuracy of N - Normal Beat: 90% (893/986)
Test Accuracy of S - Supraventricular premature or ectopic beat: 99% (999/1003)
Test Accuracy of V - Premature ventricular contraction: 95% (948/994)
Test Accuracy of F - Fusion of ventricular and normal beat: 93% (940/1001)
Test Accuracy of Q - Unclassified beat: 98% (996/1008)

Test Accuracy (Overall): 95% (4776/4992)
```

*Fig. 8: Evaluation result of model 1*

Model 1 has received 95% accuracy better than the benchmark accuracy. The model performed well on supraventricular premature or ectopic beat and unclassified beat but slightly encountered misclassification errors on other classes particularly the normal beat.

```
Test Loss: 0.123020

Test Accuracy of N - Normal Beat: 93% (921/985)
Test Accuracy of S - Supraventricular premature or ectopic beat: 99% (993/1002)
Test Accuracy of V - Premature ventricular contraction: 95% (943/989)
Test Accuracy of F - Fusion of ventricular and normal beat: 95% (960/1007)
Test Accuracy of Q - Unclassified beat: 97% (985/1009)

Test Accuracy (Overall): 96% (4802/4992)
```

*Fig. 9: Evaluation result of model 2*

As for the model 2 (fig. 9), it received much better overall accuracy of 96% compared to model 1 with 4802 out of 4992 observations correctly classified. Also, the model improved significantly on three classes that model 1 achieved slightly lower accuracy particularly in normal beat which was up by 3%. The author now saw that on the same model, careful choice of upsampling strategies is important given the fact that the latter model performed well with the same model architecture to work with.

```
Test Loss: 0.128251

Test Accuracy of N - Normal Beat: 93% (921/986)
Test Accuracy of S - Supraventricular premature or ectopic beat: 97% (973/1002)
Test Accuracy of V - Premature ventricular contraction: 95% (946/990)
Test Accuracy of F - Fusion of ventricular and normal beat: 94% (949/1006)
Test Accuracy of Q - Unclassified beat: 98% (993/1008)

Test Accuracy (Overall): 95% (4782/4992)
```

*Fig. 10: Evaluation result of model 3*

In model 3, it is evident that it received better overall accuracy compared to model 1. Although insignificant, its performance on classes like normal beat and fusion of ventricular and ectopic beat are better compared to the result of the latter model. But this used the Gaussian Noise upsampling strategy so given what accuracy we got from model 2, this may be improved.

```
Test Loss: 0.108691

Test Accuracy of N - Normal Beat: 93% (917/986)
Test Accuracy of S - Supraventricular premature or ectopic beat: 99% (991/1001)
Test Accuracy of V - Premature ventricular contraction: 96% (955/992)
Test Accuracy of F - Fusion of ventricular and normal beat: 96% (972/1006)
Test Accuracy of Q - Unclassified beat: 98% (992/1007)

Test Accuracy (Overall): 96% (4827/4992)
```

*Fig. 11: Evaluation result of model 4*

Sure enough, the accuracy and the validation/test loss for the final model with modified model architecture and a better upsampling method is better compared to the rest particularly when we look at fusion of ventricular and normal beat with highest class accuracy with test loss, almost 0.2 lower than others.

## Justification

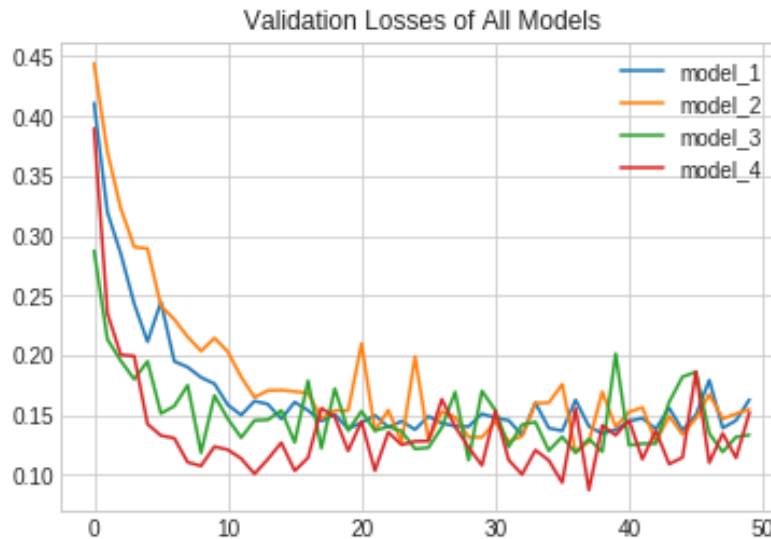
Evidently, the last model with deeper layers and higher output channels really achieved higher per-class and overall accuracies compared to the paper that only reported 93.4% of accuracy for the MIT-BIH dataset openly provided by the one of the paper's authors. Also, the appropriate choice of feature creation and upsampling strategy played an important role in increasing the performance of the model as well by comparing model\_2 and model\_4 accuracies, model\_4 being the best of the two.

Finally, choice of hyperparameters like the batch size, and the number of epochs and the learning rate also contributed to the increased performance of the final model. All of these are fulfilled using PyTorch framework which proved to be a viable option to achieve end-to-end reproducible deep learning projects like this.

## V. Conclusion

### Free-Form Visualization

After training all of the models, the author retrieved each of the recorded validation losses for 50 epochs per model and compare their progressions via line graph. As you can see all of the models except model 4 have reached convergence around epoch 20. Specifically, at epoch 39, **model\_4** has achieved its lowest validation loss (approximately 0.08).



*Fig. 12: Validation Losses of all Models Trained*

### Reflection

There are numerous Kaggle kernel submissions for this particular dataset that attempted to either replicate the results from the paper or create new ones via TensorFlow which the paper used as well as Keras and scikit-learn frameworks. As of this writing, no one has attempted to create models from the open dataset using PyTorch framework<sup>19</sup>. This therefore is the first project that uses the same dataset implemented using PyTorch.

Also, the unbalanced nature of the dataset that the author used in this project lead to his exploration of other means to augment classes with lower samples. To come up with two effective strategies to do so and be able to implement that in the whole pipeline is also something that the author is compelled to be proud of.

Finally, an accuracy of 93.4 for MIT-BIH dataset is hard to beat. Being able to achieve higher than the benchmark accuracy for all four models exceeded author's expectation of the project's output.

---

<sup>19</sup> Fazeli, Shayan. "ECG Heartbeat Categorization Dataset Kaggle Kernels." Kaggle, 31 May 2018, [www.kaggle.com/shayanfazeli/heartbeat/kernels](https://www.kaggle.com/shayanfazeli/heartbeat/kernels).

## **Improvement**

To achieve an end-to-end deep learning pipeline for this project, the author has yet to create a service that would take up digitized EKG readings and output a certain heartbeat class mentioned in this document. Also, it is also important to remember that the basis for the dataset came from manually-digitized EKG signals on a fixed sampling rate. Aside from an actual data product, it will also be beneficial for the ones in need to use this tool to automatically read certain EKG outputs and convert them to data structures similar to the inputs we utilized in this project.