SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

# Sleep Apnea Detection

## 50.039 Deep Learning Project Report

Edwin Wongso 1006200
Wongsaphat Pipatsawettanan 1006297
Lim Yongqing 1005955

## 1. Introduction and Problem

Sleep apnoea/apnea is a disorder that causes a person to stop breathing while asleep. The brain tries to protect you by waking them up enough to breathe, but this prevents restful, healthy sleep. Over time, this condition can cause complications such as depression, quick-temperedness and heart-related problems. Therefore, our primary objective is to enhance the efficacy of hospitals and clinics in the prediction of periods characterized by sleeping disorders, utilizing electrocardiogram (ECG) data obtained from patients.  (Refer to Readme to run code)

# 2. Dataset preprocessing

## 2.1 Brief description of dataset

The dataset [2] that we are looking at are records of ECG readings of patients experiencing Apnea and non-Apnea episodes, each recording's total duration ranges from 7 to 10 hours each. Each recording includes primarily, a continuous digitized ECG signal and a set of apnea annotations. The apnea annotations are "A" for apnea and "N" for normal.

## 2.2 Train-Test Split

The dataset consists of a total of 70 such sets of recording, already split into 35 for the training set and the other 35 for the testing set. Each 60 second segment is labeled with either an 'A' to denote if Apnea is present during the segment or 'N' otherwise.

It is also important to note that the distributions of the training files are not even. For example, 20 training samples starting with 'a' have more 'A' tags than the rest, and 10 training sample files starting with 'c' having the least number (almost no number) of 'A' tags. The final 5 training sample files starting with 'b' are roughly in the middle with more 'A' tags than 'c' but less than 'a'. Given this file composition, When the files are combined, we see a fairly balanced distribution of "N" and "A" tags.

All of the models we tried will conduct per segment classification, therefore we can shuffle the samples across files in the train set.

## 2.3 Pre-processing

### 2.3.1: Parsing Dataset Files

To parse the dataset file, we used the `wfdb` python library for reading the ECG signal files. It allows us to read annotation and ECG signal files as numpy arrays, which we will process further. Details of preprocessing are in the following sections.

### 2.3.2: Filtering of Signal

First, we apply a filter to the readings so as to remove any unnecessary noise from the data so that our model can converge better. Popular choices of filter include the Chebyshev or Butterworth filter, commonly used for signal processing. We decided to use the Chebyshev filter due to the nature of our ECG reading, as there are sharp transitions between the pass band and the stop band of the signals. Furthermore, butterworth filters would generate smoother and more consistent signals which may affect the identification of Apnea scenarios. Thus, we use the Chebyshev filter with a cutoff frequency of 48Hz [1] on the inputs, which are then passed as new inputs into the model.

### 2.3.3: Spectrogram Creation and Segmentation

For each audio file, we created the spectrogram for the entire signal of a single recording. We created the spectrogram with parameters:

NFFT = 128,
noverlap = 16,
Sample frequency = 100 (as specified by dataset)

We tried multiple different values of NFFT: 128, 256, 512, and it did not make a significant difference to our model's results, so we chose NFFT = 128 which should be a sufficient number of frequency bands for our signal.

We used the `matplotlib.pyplot.specgram` method to create a spectrogram numpy array.

Within our `SpectrogramDataset` object we have a function called `get_spectrogram_and_annotations` to align each annotation to the 60s segment in the spectrogram corresponding to it, such that the start index and end index for each spectrogram segment has a start and end time which are as close to the corresponding 60s segment as possible.

# 3. Model Architecture

We tried 3 different models for this classification task, all of them work on per-segment classification, classifying each spectrogram segment as either 1 for “A” or 0 for “N”. The three models we tried were Pure CNN, Pure LSTM and CNN-LSTM.

Define

$$b = batch\ size,\ c\ =\ number\ of\ channels,\ h\ =\ height,\ w\ = width$$

## 3.1 Pure Convolutional Neural Network (CNN)

In the pure CNN model, we pass the segment spectrogram of shape $(b, c_{in}, h_{in}, w_{in})$ into a CNN. We then reshape the CNN output tensor to the shape $(w_{out}, b, c_{out} * h_{out})$. This reshaped output tensor is then passed into a sequential layer consisting of 3 linear layers with ReLU and a sigmoid at the end.

| Layer | Type |
|---|---|
| 1 | Convolution, kernel = (2,2), filters = 128 |
| 2 | Batch Normalisation |
| 3 | ReLU |
| 4 | Convolution, kernel = (2,2), filters = 128 |
| 5 | Batch Normalisation |
| 6 | ReLU |
| 7 | Max Pooling, window = (2,2) |
| 8 | Convolution, kernel = (2,2), filters = 128 |

| | |
|---|---|
| 9 | Batch Normalisation |
| 10 | ReLU |
| 11 | Max Pooling, window = (2,2) |
| 12 | Convolution, kernel = (2,2), filters = 128 |
| 13 | Batch Normalisation |
| 14 | Reshape CNN output: $(b, c_{out}, h_{out}, w_{out})$ ->$(w_{out}, b, c_{out} * h_{out})$ |
| 15 | Fully Connected Layer, input_size = 67712, output_size = 1 |
| 16 | Sigmoid |

## 3.2 Pure Long-term Short-term Memory (LSTM)

The pure LSTM model architecture comprises of an LSTM layer, followed by either a linear layer or a batch normalization layer or no additional layers (FILL), then a leaky Relu, and finally a sequential layer comprising of another linear layer with a sigmoid at the end, given below:

(Below values are default settings)

| Layer | Type |
|---|---|
| 1 | LSTM, input_size = 32*7*7, hidden_size = 512, num_layers = 1, batch_first = False |
| 2 | Dropout, dropout = 0.4 |
| 3 | (FILL) |
| 4 | LeakyReLU, alpha = 0.01 |
| 5 | Fully Connected Layer, input_size = 256, output_size = 1 |

| | |
|---|---|
| 6 | Sigmoid |

Additional layer goes into (FILL):

| Layer | Type |
|---|---|
| 1 | Fully Connected Layer, input_size = 512, output_size = 256 |
| 2 | BatchNorm1D, num_features = 1 |
| 3 | NIL (No additional layer) |

*The different parameters used will be highlighted below in the hyper parameter section.

## 3.3 CNN-LSTM

In this model, we pass the segment spectrogram of shape $(b, c_{in}, h_{in}, w_{in})$, into a CNN which produces an output of shape $(b, c_{out}, h_{out}, w_{out})$. After the CNN forward pass, we reshape the output tensor from the CNN to the shape $(w_{out}, b, c_{out} * h_{out})$.
We pass this output to the LSTM where $w_{out}$ acts as the sequence length and $c_{out} * h_{out}$ acts as the input dimension to the LSTM. After the forward pass to the LSTM, we take the final output, $y_{w_{out}}$ ,of the LSTM and use that input to a fully connected layer to classify the segment as either indicating “A” or “N” for apnea or normal.

The rationale behind this model is that the CNN will be able to detect common features between spectrograms for samples of the same class, and the LSTM will be able to treat the output of the CNN as a sequence along the time domain, and the last output in the LSTM’s output sequence will have used information from previous time steps.

| Layer | Type |
|---|---|
| 1 | Convolution, kernel = (2,2), filters = 128 |
| 2 | Batch Normalisation |
| 3 | ReLU |
| 4 | Convolution, kernel = (2,2), filters = 128 |
| 5 | Batch Normalisation |
| 6 | ReLU |
| 7 | Max pooling, window = (2,2) |
| 8 | Convolution, kernel = (2,2), filters = 128 |
| 9 | Batch Normalisation |

| | |
|---|---|
| 10 | ReLU |
| 11 | Convolution, kernel = (2,2), filters = 128 |
| 12 | Reshape CNN output: $(b, c_{out}, h_{out}, w_{out})$ ->$(w_{out}, b, c_{out} * h_{out})$ |
| 13 | Batch Normalisation |
| 14 | ReLU |
| 15 | LSTM, input size = $c_{out} * h_{out}$ hidden size = 512, 3 layers |
| 16 | Fully Connected Layer, input size = 512, output size = 1 |

# 4. Training

## 4.1 Loss function

Given that this is a binary classification task, we used torch's binary cross entropy loss for each sample, which has the following definition.

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log(1 - \hat{Y}_i))$$

For our optimizer we chose the Adam optimizer.

## 4.2 Metrics

We recorded loss, precision, recall, accuracy during each train and test epoch in our code.

Furthermore, we decided to calculate the following composite score,

$$Composite\ of\ accuracy\ and\ f1 = 0.4 * \ accuracy \ + \ 0.6 * f1$$

We calculate the composite scores based on accuracy, precision and recall in order to choose the best model for generalization. However, it is difficult to assess which one is the best overall based on performance on each specific metric, and thus, consider their weighted sum instead. Having stated above that the dataset is not similarly distributed, we decide to assign a lower weight to accuracy, and combine precision and recall to form the f1 score, and assign a higher weight to the f1 score, which is more ideal for class imbalances. This would also allow us to choose models that are better in classifying positive scenarios and not misclassifying positive cases.

# 5. Experimentation

In this section, we tried modifying each model's

1. architecture
2. hyperparameters

and making a comparison between different configurations for each model and finding the optimal result based on our composite of accuracy and F1.

## 5.1 Pure CNN

Hyperparameters tuning

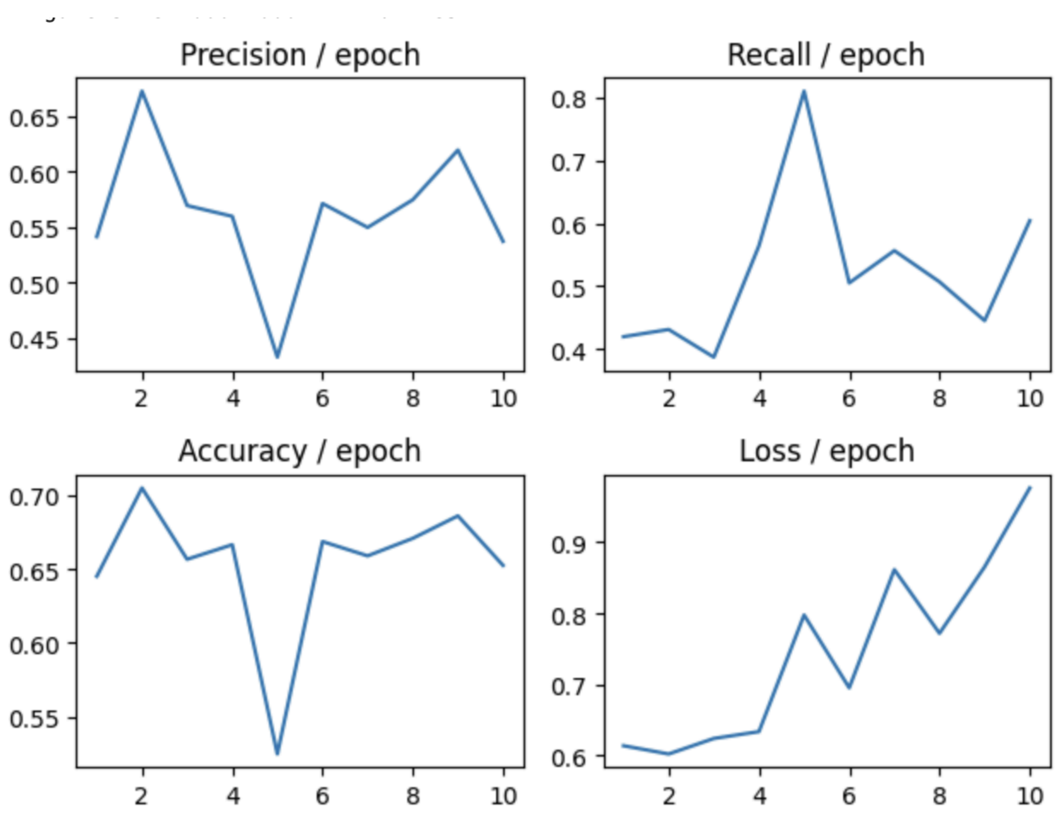
| Model versions | Batch size | Dropout value | Loss Function | Learning Rate |
|---|---|---|---|---|
| v1 | 1 | 0.2 | BinaryCrossEntropyLoss | 0.001 |
| v2 | 8 | 0.2 | BinaryCrossEntropyLoss | 0.001 |
| v3 | 8 | 0.4 | BinaryCrossEntropyLoss | 0.001 |
| v4 | 8 | 0.4 | BinaryCrossEntropyLoss | 0.0001 |
| v5 | 8 | 0.5 | BinaryCrossEntropyLoss | 0.001 |
| v6 | 8 | 0.5 | BinaryCrossEntropyLoss | 0.0001 |
| v7 | 8 | 0.1 | BinaryCrossEntropyLoss | 0.001 |
| v8 | 8 | 0.1 | BinaryCrossEntropyLoss | 0.0001 |

(Values are based on final value; Best scores for each column are bolded)

| Model versions | Final loss | Accuracy | Precision | Recall | F1 Score | Composite of Accuracy and F1 |
|---|---|---|---|---|---|---|
| v1 | 6.36875 | 0.62555 | 0.50639 | 0.50809 | 0.50723 | 0.55456 |
| v2 | 0.80755 | 0.63018 | 0.51154 | 0.55450 | 0.53215 | 0.57136 |
| v3 | 0.97641 | 0.65236 | 0.53711 | **0.60427** | **0.56871** | **0.60217** |
| v4 | 1.16076 | 0.65971 | **0.56336** | 0.45740 | 0.50488 | 0.56681 |
| v5 | **0.80386** | 0.65664 | 0.55074 | 0.51450 | 0.53200 | 0.58186 |
| v6 | 0.96404 | 0.60510 | 0.48354 | 0.60351 | 0.53690 | 0.56418 |
| v7 | 1.09867 | 0.65265 | 0.55542 | 0.42229 | 0.47979 | 0.54893 |
| v8 | 0.88777 | **0.66191** | 0.55801 | 0.52274 | 0.50238 | 0.56619 |

## 5.1.1 Graph of PureCNN Best Result (V3)

Test Set Metrics for PureCNN V3 over 10 epochs



## 5.2 Pure LSTM

### 5.2.1 Pure LSTM Versions

For reference, the general LSTM model architecture used is:

```
Class PureLSTM(nn.Modules):
   def __init__(self, dropout = 0.2):
      self.rnn = nn.LSTM(65, 512, 2, dropout = dropout)
      self.rnn = FILL
      self.relu = nn.LeakyReLU()
      self.classifier = nn.Sequential(
         nn.Linear(256, 1),
         nn.Sigmoid(),
      )
   def forward(self, src):
      …
```

In **FILL**, one additional layer is added here, which includes either a linear layer, batch norm layer, or no layers.

### 5.2.2 Pure LSTM Versions

The table below shows the hyper parameters used for 6 slightly different LSTM models:

| Model versions | Dropout value | LSTM layer dimensions | Includes batchnorm | Includes additional linear layer | Layer dims (**FILL = nn.Linear()**) | LeakyReLU alpha value |
|---|---|---|---|---|---|---|
| v1 | 0.4 | (65, 512, 2) | No | No | N.A. | 0.01 |
| v2 | 0.5 | (65, 512, 2) | No | No | N.A. | 0.01 |
| v3 | 0.5 | (65, 512, 2) | Yes | No | N.A. | 0.01 |
| v4 | 0.4 | (65, 512, 2) | No | Yes | (512, 128) | 0.01 |
| v5 | 0.4 | (65, 512, 2) | No | Yes | (512, 256) | 0.01 |

| | | | | | | |
|---|---|---|---|---|---|---|
| v6 | 0.4 | (65, 512, 2) | No | No | N.A. | 0.1 |
| v7* | 0.4 | (100, 512, 2) | No | No | N.A. | 0.1 |
| v8* | 0.4 | (100, 256, 2) | No | No | N.A. | 0.01 |
| v9* | 0.5 | (100, 128, 2) | Yes | No | N.A. | 0.05 |

*Has additional padding to ensure that input size is constant

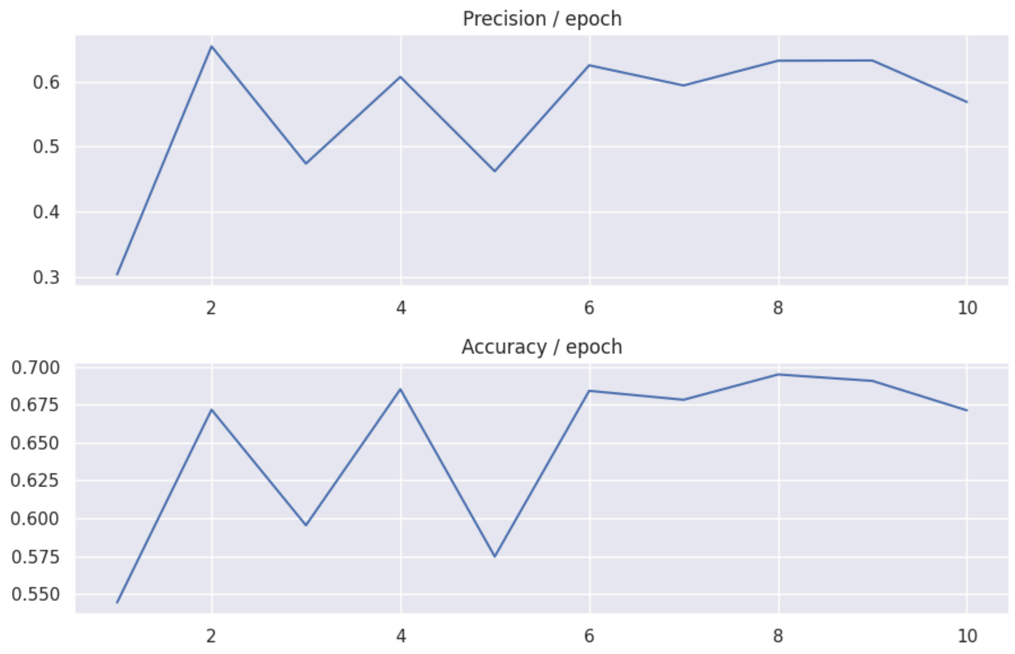
(Values are based on final value; Best scores for each column are bolded)

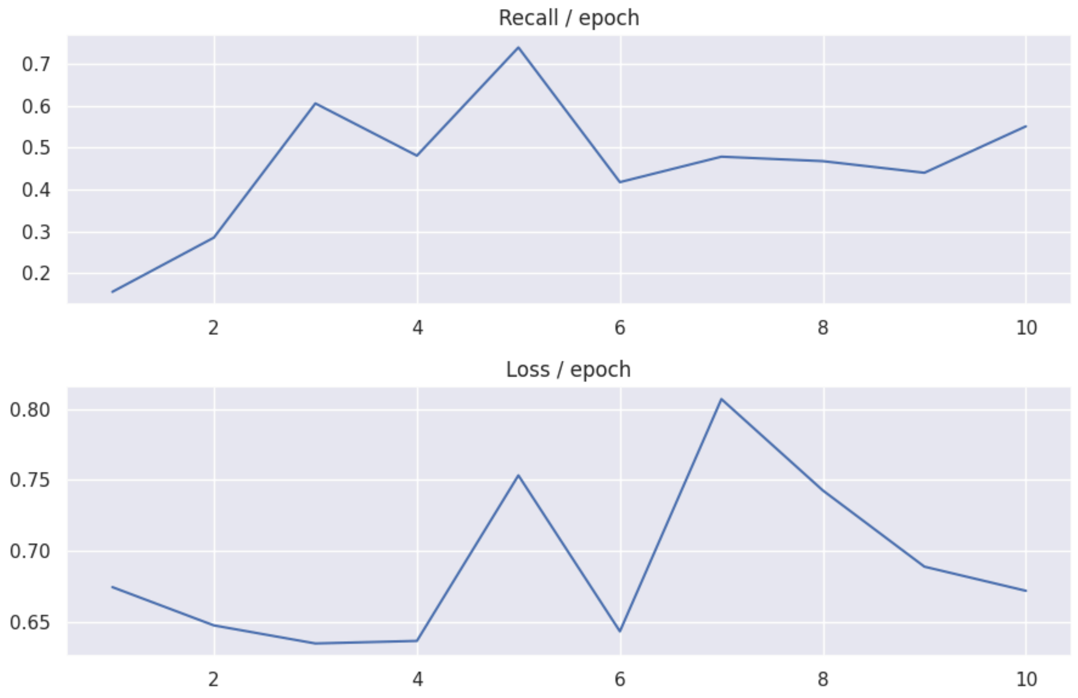
| Model versions | Final loss | Accuracy | Precision | Recall | F1 score | Composite of Accuracy and F1 |
|---|---|---|---|---|---|---|
| v1 | 0.73292 | 0.66609 | 0.57174 | 0.47695 | 0.52006 | 0.57847 |
| v2 | 0.70616 | 0.58553 | 0.46918 | **0.70534** | 0.56352 | 0.57232 |
| v3 | **0.61430** | 0.68167 | 0.59936 | 0.48489 | 0.53608 | 0.59432 |
| v4 | 0.70672 | 0.67964 | 0.60209 | 0.45832 | 0.52046 | 0.58413 |
| v5 | 0.67190 | 0.67124 | 0.56875 | 0.55130 | 0.55989 | **0.60443** |
| v6 | 0.72626 | **0.69446** | 0.64911 | 0.42336 | 0.51248 | 0.58527 |
| v7* | 0.69560 | 0.65543 | **0.68496** | 0.16962 | 0.27191 | 0.42532 |
| v8* | 0.67071 | 0.64362 | 0.52548 | 0.62351 | **0.57031** | 0.599634 |
| v9* | 0.64557 | 0.65937 | 0.55347 | 0.52779 | 0.54033 | 0.58795 |

*Has additional padding to ensure that input size is constant

### 5.2.3 Graph of PureLSTM Best Result (V5)

<u>Test Set Metrics for PureLSTM V5 over 10 epochs.</u>





## 5.3 CNN-LSTM

### 5.3.1 CNN-LSTM Versions

| Model Version | Convolution Number of Filters/Layer | Number LSTM Hidden Layers |
|---|---|---|
| V1 | 32 | 1 |
| V2 | 64 | 1 |
| V3 | 128 | 1 |
| V4 | 128 | 2 |
| V5 | 64 | 2 |
| V6 | 32 | 2 |

### 5.3.2 CNN-LSTM Results

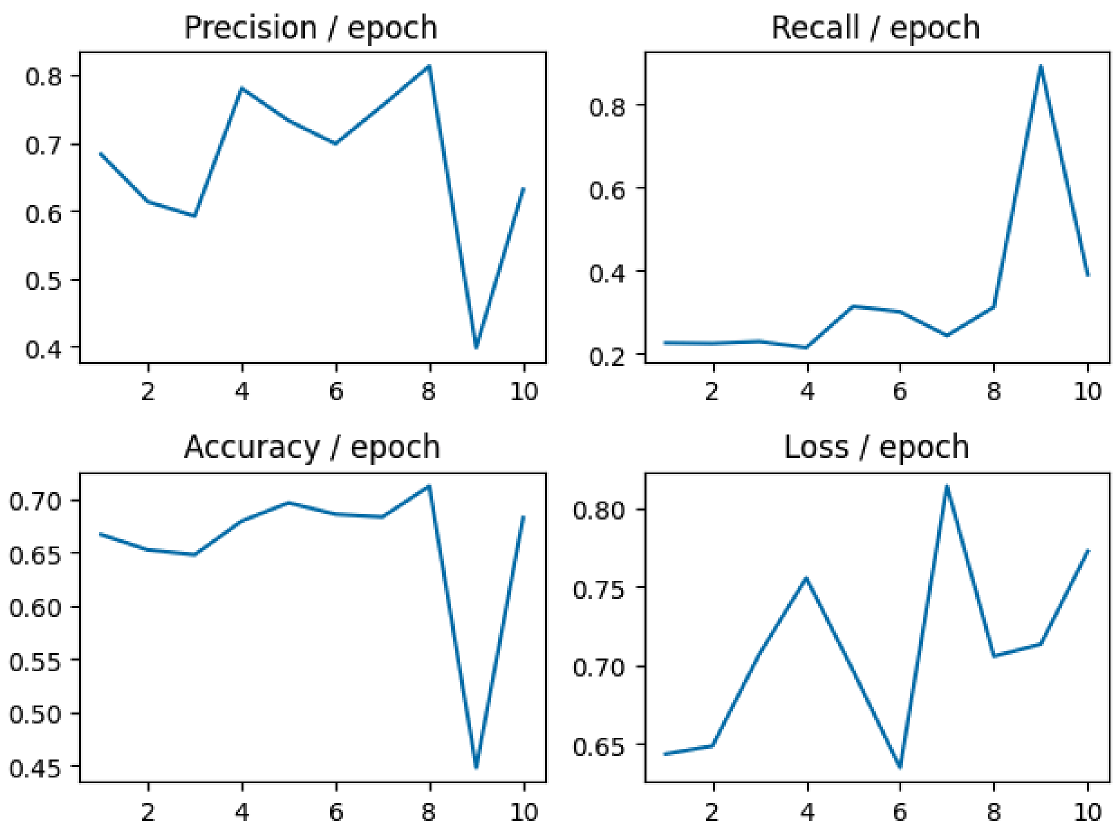
| Model versions | Final loss | Accuracy | Precision | Recall | F1 Score | Accuracy and F1 Composite |
|---|---|---|---|---|---|---|
| v1 | **0.67269** | **0.69458** | **0.77787** | 0.27267 | 0.40380 | 0.5201 |
| v2 | 0.75362 | 0.65514 | 0.70877 | 0.15419 | 0.2528 | 0.41374 |
| v3 | 3.49798 | 0.59069 | 0.59069 | **0.45145** | **0.51177** | 0.54333 |
| v4 | 0.77255 | 0.68236 | 0.63131 | 0.39083 | 0.48278 | **0.56261** |

| | | | | | | |
|---|---|---|---|---|---|---|
| v5 | 0.76097 | 0.61831 | 0.49597 | 0.38610 | 0.43419 | 0.50784 |
| v6 | 0.84121 | 0.63840 | 0.53435 | 0.36335 | 0.43256 | 0.5159 |

As shown in the table above, v4 gives the best composite score of Accuracy and F1-score, however one common trend across all versions is that recall is significantly lower than accuracy and  precision.

### 5.3.3 Graph of CNN-LSTM Best Result (V5)

Test Set Metrics for CNN-LSTM V4 over 10 epochs.



# 6. Comparison with existing models

## 6.1 Differences in implementations

We refer to the research done by Asghar Zarei a, Hossein Beheshti b, Babak Mohammadzadeh Asl [1] on Detection of sleep apnea using deep neural networks and single-lead ECG signal as a starter for our project. There are various differences between the two implementations.

First being the dataset. The research project includes two different datasets which are Apnea-ECG and UCDDB datasets while our project works with just the Apnea-ECG dataset.

The second difference is our spectrograms are likely different, as their research paper never indicated the parameters used to create the spectrogram they use as input to their CNN-LSTM model.

## 6.2 Differences in results

Overall, their results seem more promising, as their accuracy was 97.21% whereas the highest accuracy we obtained was about 69.4% using our PureLSTM model.

Our implementation of CNN-LSTM, which is the architecture that the research paper used[1] , was unable to match their results, likely due to the way we create our 2D signal, as [1] refers to using a 2D signal as input but never explicitly stating the parameters used for their 2D signal.

In addition, the model architecture the paper used is not released and thus, there is no way of knowing what layers were used nor how many parameters their model contains. Due to this, there could be a gap between model size which hinders the ability of our model to generalize well. In addition, hardware limitations also hinder our ability to train a bigger model, based on multiple attempts at running models with more layers.

Lastly, we emphasize more on recall and precision as well. Unlike the paper which only reported accuracy scores, it would be difficult to gauge the extent of the ability of our model without external benchmarking.

## 7. Conclusion and Future Works

The three models we tested did not seem to match the cutting edge in this field. In terms of future works, a vision transformer may be able to yield high accuracy and f1-score for this particular classification task, wherein the input is each 60s spectrogram segment.

Another aspect we could possibly try is to consider multi-segment classification, wherein when classifying one segment, we could look at other segments surrounding it. This may improve the model's ability to retain information from the previous few segments to improve its ability to predict apnea episodes.

## References:

[1] A. Zarei, H. Beheshti, and B. M. Asl, “Detection of sleep apnea using deep neural networks and single-lead ECG Signals,” Biomedical Signal Processing and Control, https://www.sciencedirect.com/science/article/pii/S1746809421007229?ref=pdf_download&fr=RR-2&rr=871a096dfc33a1ad (accessed Apr. 14, 2024).

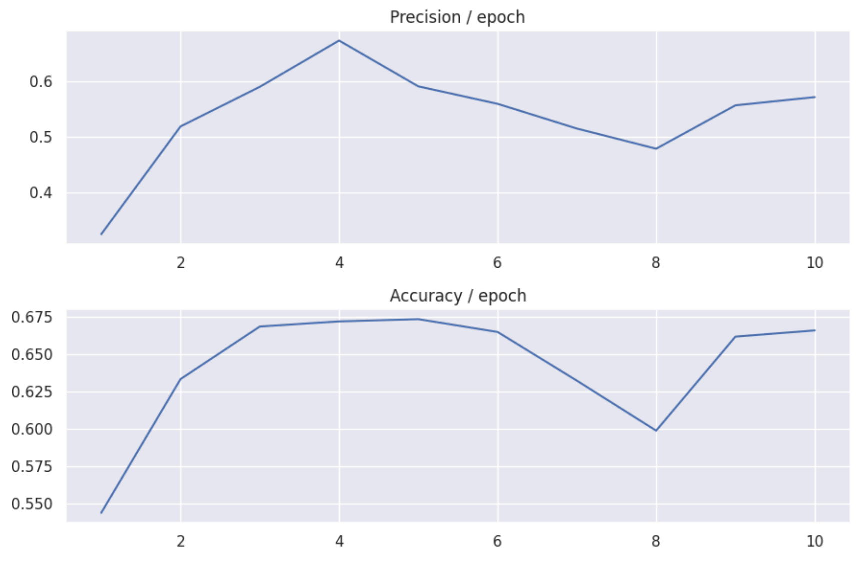
[2] G. Moody and R. Mark, “Apnea-ECG database,” Apnea-ECG Database v1.0.0, https://physionet.org/content/apnea-ecg/1.0.0/ (accessed Apr. 14, 2024).

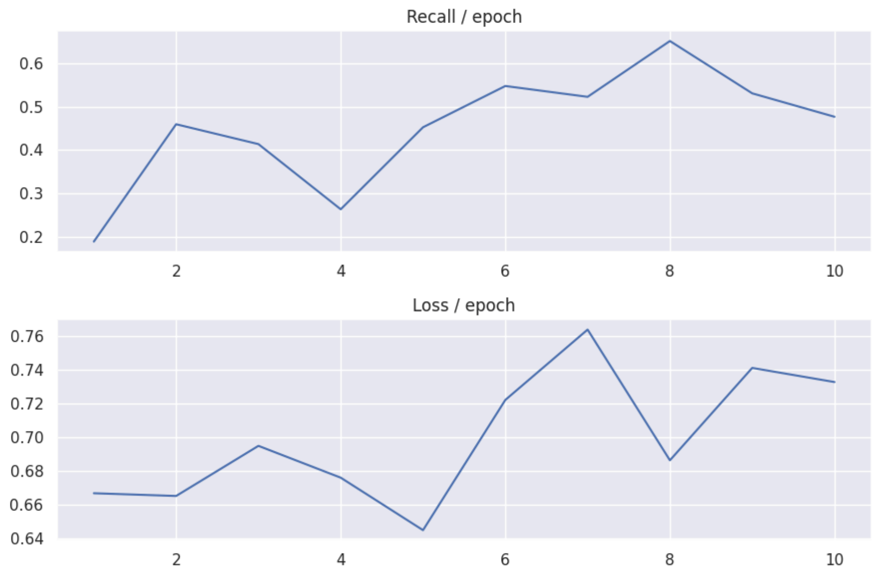
## Appendix:

Graphs for all model versions test set metrics

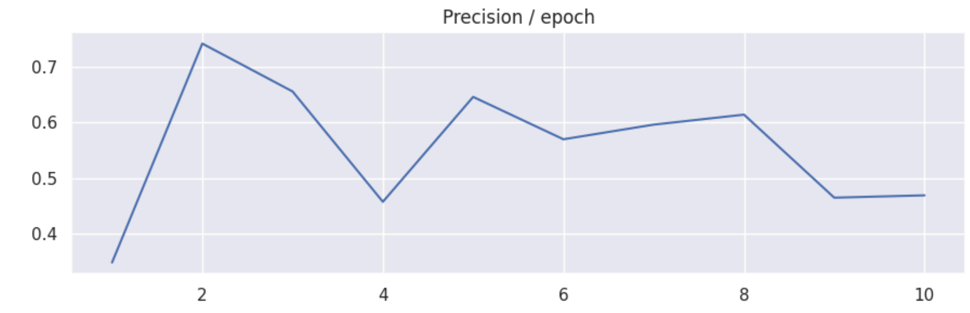
Pure LSTM:

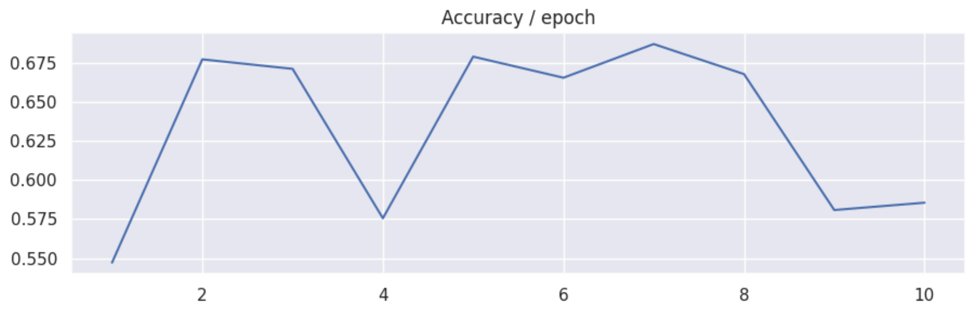
PureLSTM V1

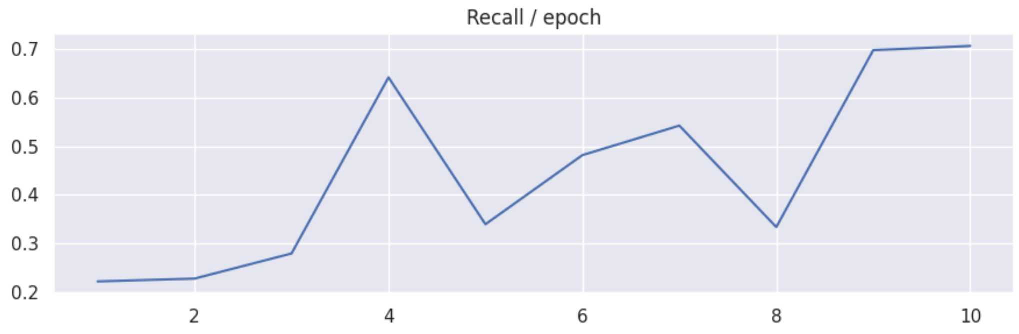


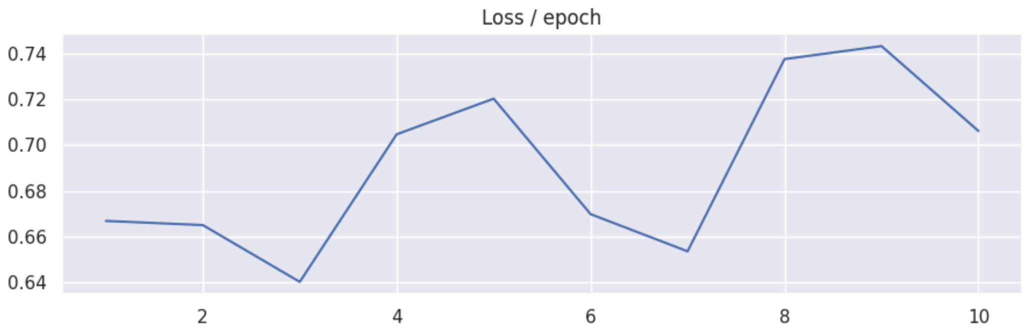


## PureLSTM V2

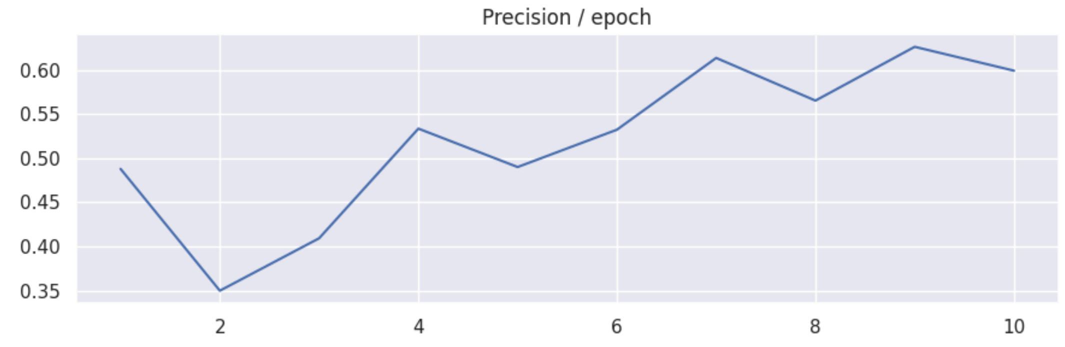

Precision / epoch
0.7
0.6
0.5
0.4
2
4
6
8
10

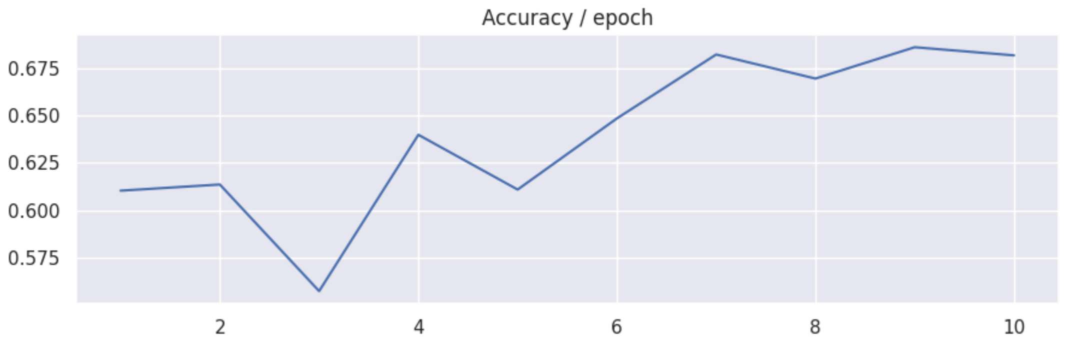

Accuracy / epoch
0.675
0.650
0.625
0.600
0.575
0.550
2
4
6
8
10

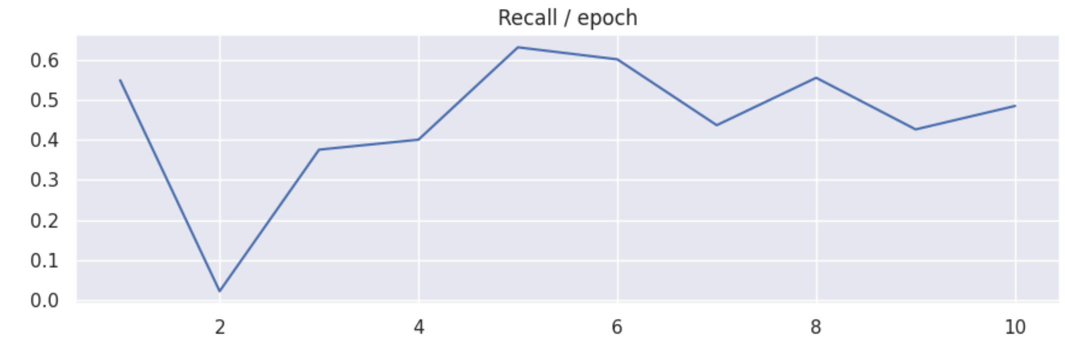

Recall / epoch
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10

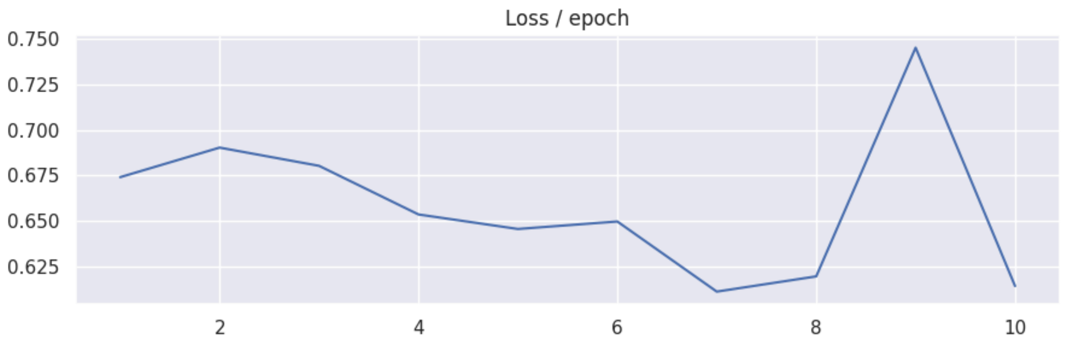

Loss / epoch
0.74
0.72
0.70
0.68
0.66
0.64
2
4
6
8
10

# PureLSTM V3

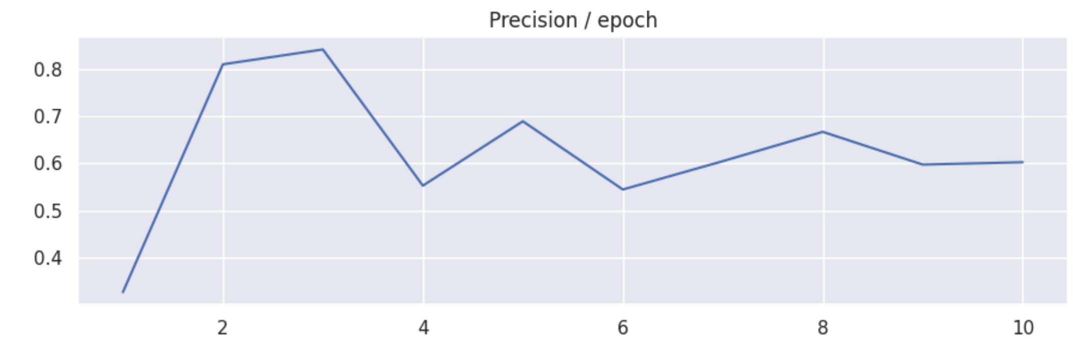

Precision / epoch
0.60
0.55
0.50
0.45
0.40
0.35
2
4
6
8
10

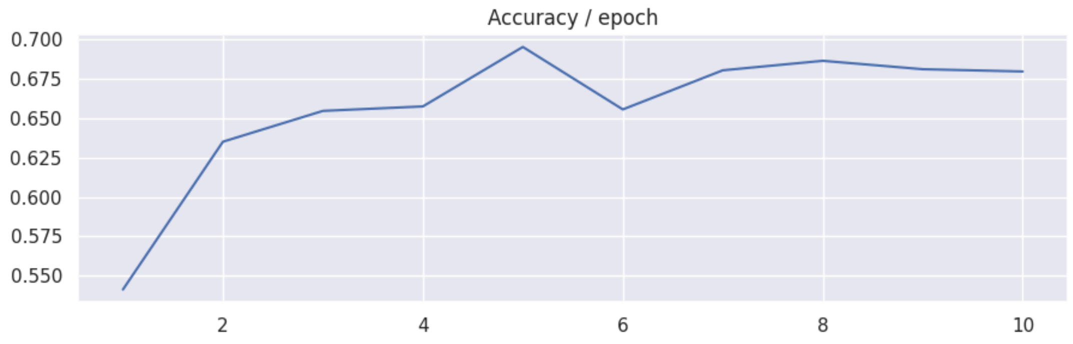

Accuracy / epoch
0.675
0.650
0.625
0.600
0.575
2
4
6
8
10

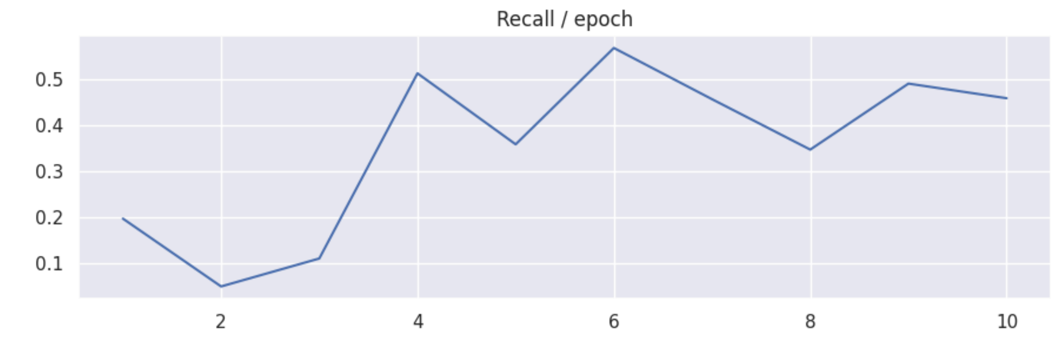

Recall / epoch
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2
4
6
8
10

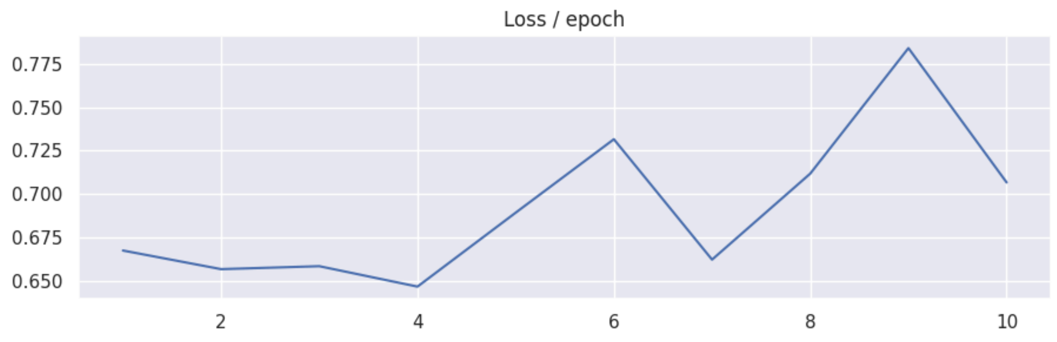

Loss / epoch
0.750
0.725
0.700
0.675
0.650
0.625
2
4
6
8
10

# PureLSTM V4

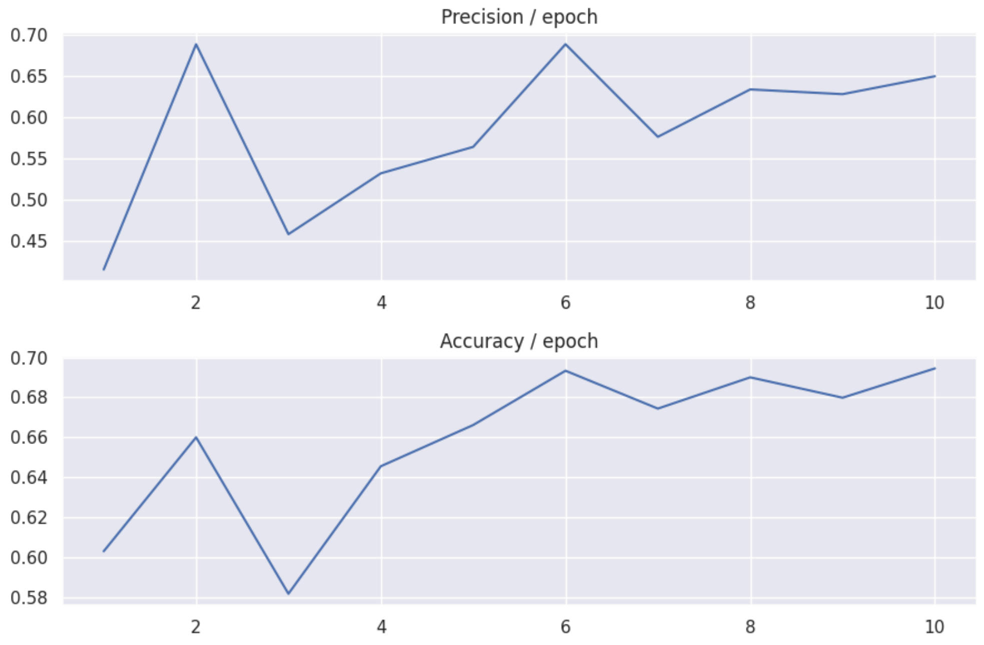

Precision / epoch
0.8
0.7
0.6
0.5
0.4
2
4
6
8
10

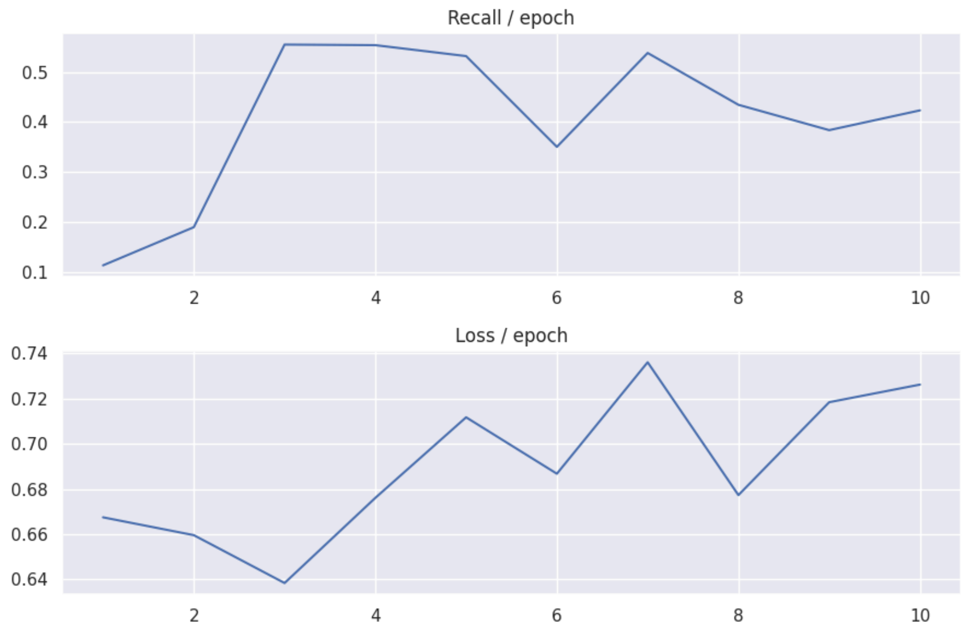

Accuracy / epoch
0.700
0.675
0.650
0.625
0.600
0.575
0.550
2
4
6
8
10


Recall / epoch
0.5
0.4
0.3
0.2
0.1
2
4
6
8
10

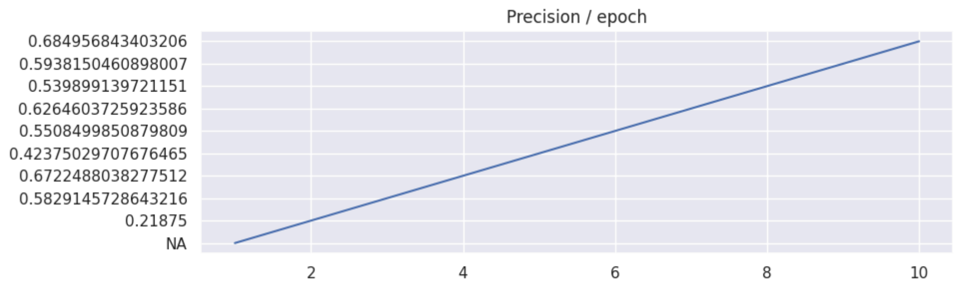

Loss / epoch
0.775
0.750
0.725
0.700
0.675
0.650
2
4
6
8
10

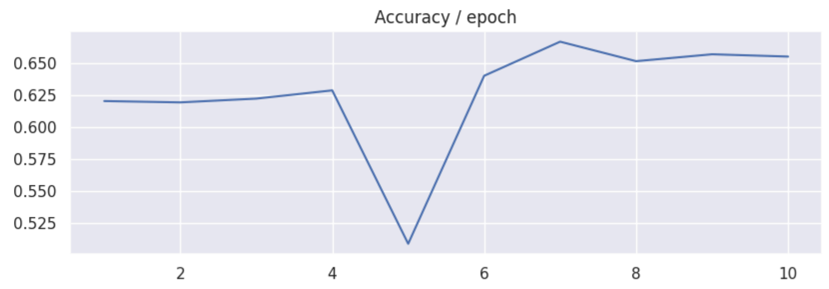
## PureLSTM V6

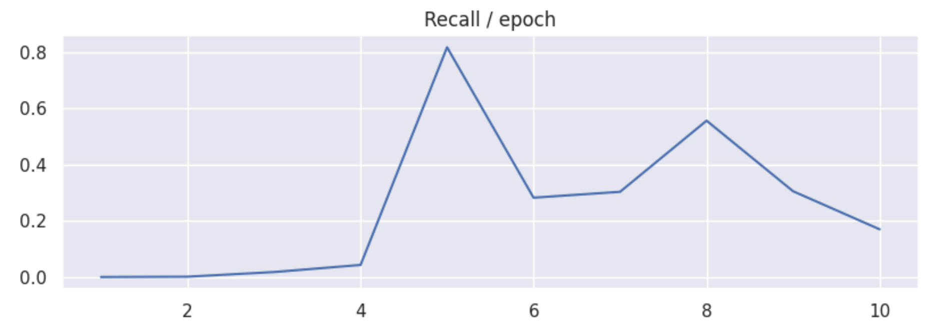

Precision / epoch
0.70
0.65
0.60
0.55
0.50
0.45
2
4
6
8
10
Accuracy / epoch
0.70
0.68
0.66
0.64
0.62
0.60
0.58
2
4
6
8
10

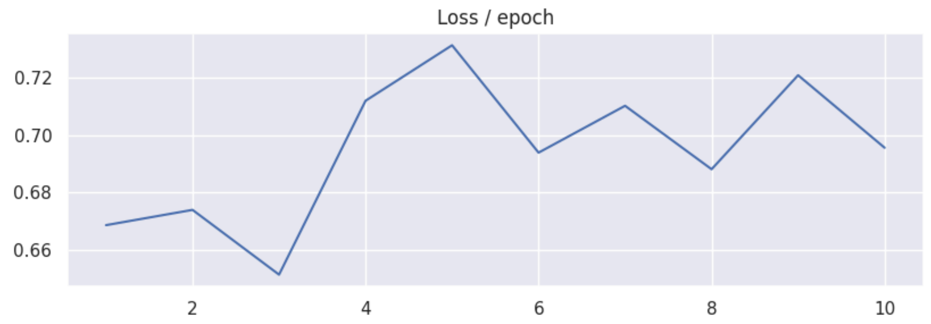

Recall / epoch
0.5
0.4
0.3
0.2
0.1
2
4
6
8
10
Loss / epoch
0.74
0.72
0.70
0.68
0.66
0.64
2
4
6
8
10

# PureLSTM V7

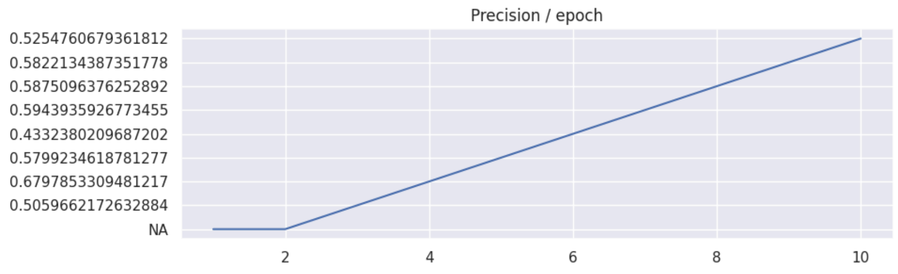

Precision / epoch
0.684956843403206
0.5938150460898007
0.539899139721151
0.6264603725923586
0.5508499850879809
0.42375029707676465
0.6722488038277512
0.5829145728643216
0.21875
NA
2
4
6
8
10

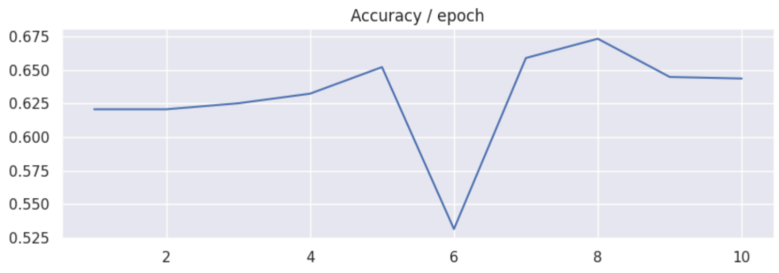

Accuracy / epoch
0.650
0.625
0.600
0.575
0.550
0.525
2
4
6
8
10

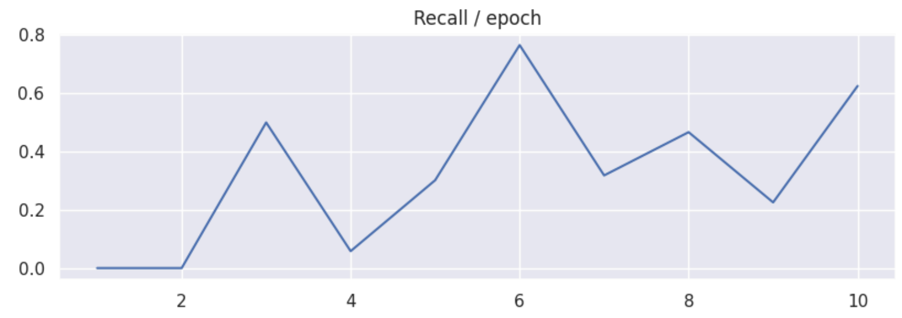

Recall / epoch
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10

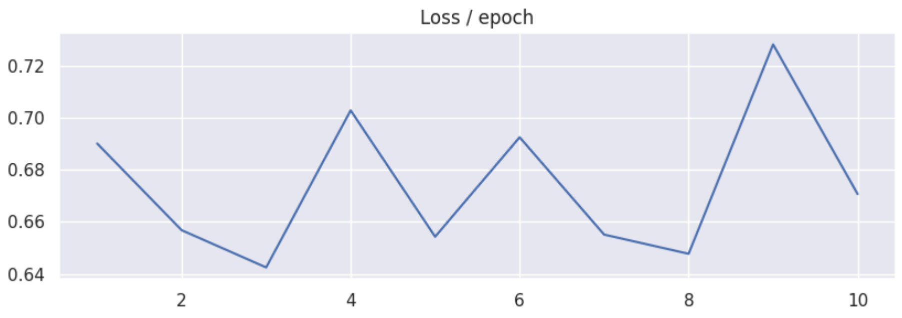

Loss / epoch
0.72
0.70
0.68
0.66
2
4
6
8
10

# PureLSTM V8

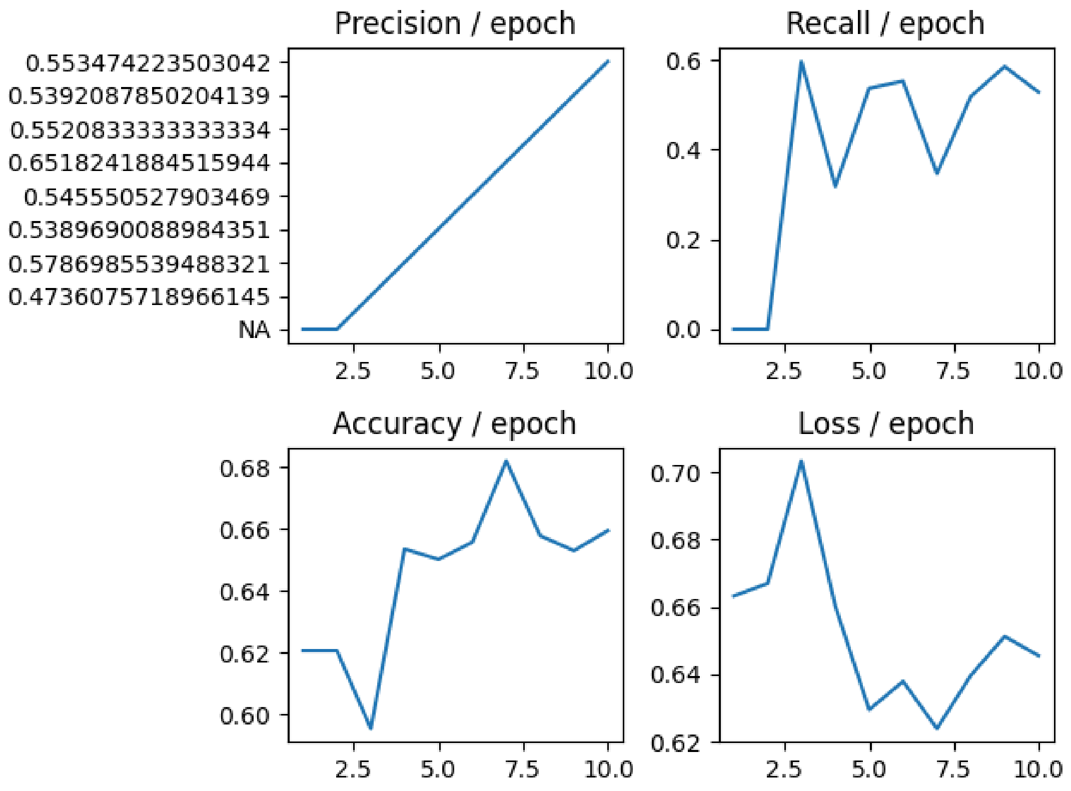

Precision / epoch
0.5254760679361812
0.5822134387351778
0.5875096376252892
0.5943935926773455
0.4332380209687202
0.5799234618781277
0.6797853309481217
0.5059662172632884
NA
2
4
6
8
10


Accuracy / epoch
0.675
0.650
0.625
0.600
0.575
0.550
0.525
2
4
6
8
10


Recall / epoch
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10


Loss / epoch
0.72
0.70
0.68
0.66
0.64
2
4
6
8
10

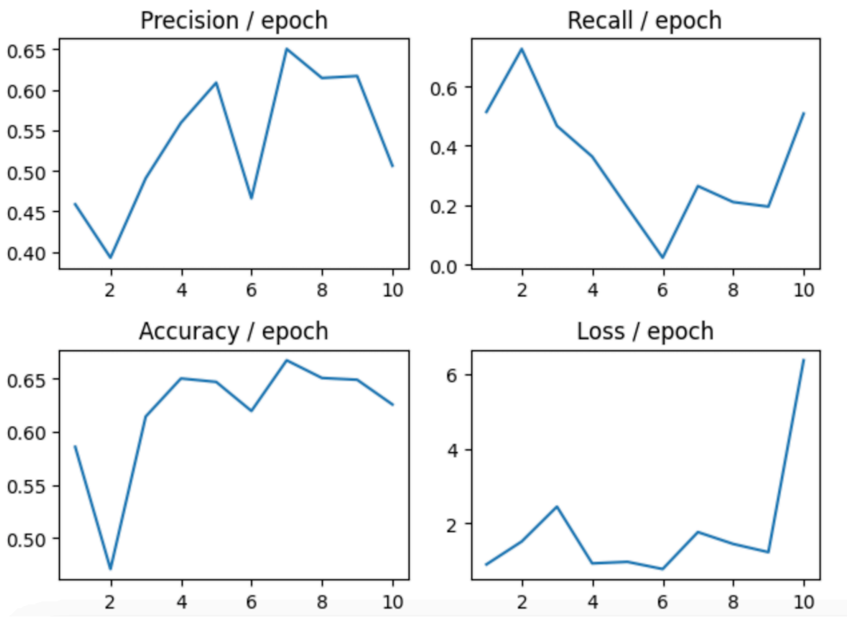
PureLSTM V9



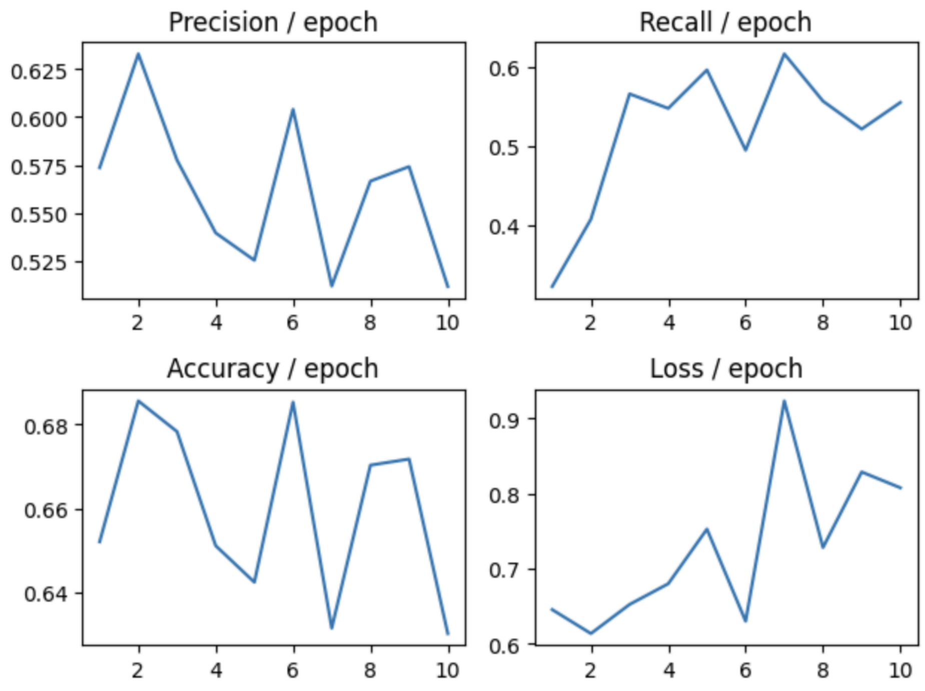
Precision / epoch
0.553474223503042
0.5392087850204139
0.5520833333333334
0.6518241884515944
0.545550527903469
0.5389690088984351
0.5786985539488321
0.4736075718966145
NA
2.5
5.0
7.5
10.0
Recall / epoch
0.6
0.4
0.2
0.0
Accuracy / epoch
0.68
0.66
0.64
0.62
0.60
Loss / epoch
0.70
0.68
0.66
0.64
0.62

Pure CNN:

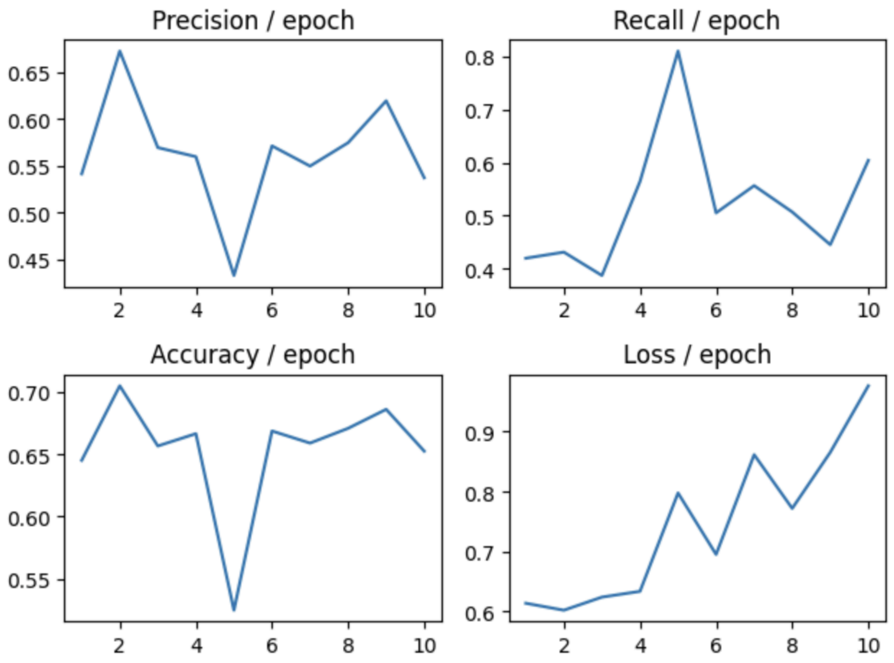
PureCNN_V1



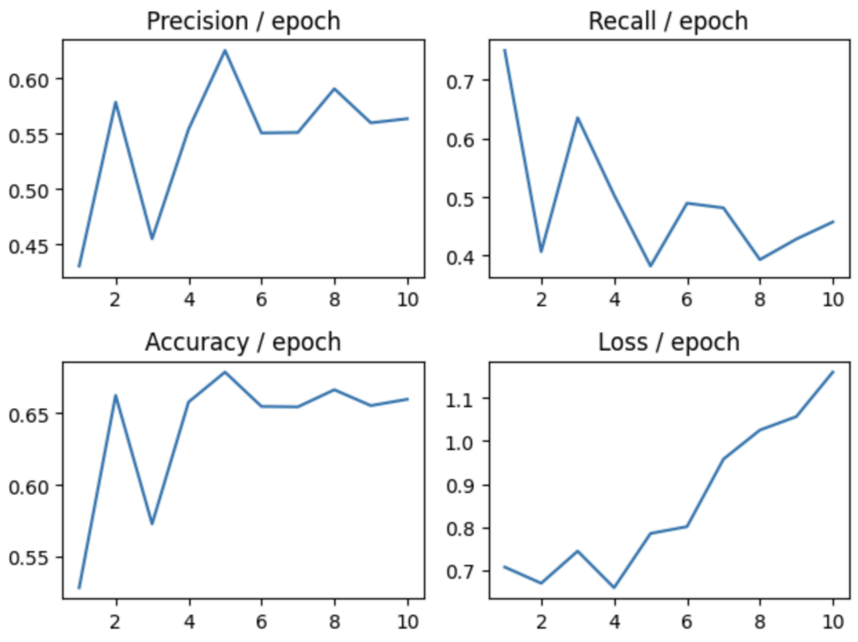
PureCNN_V2



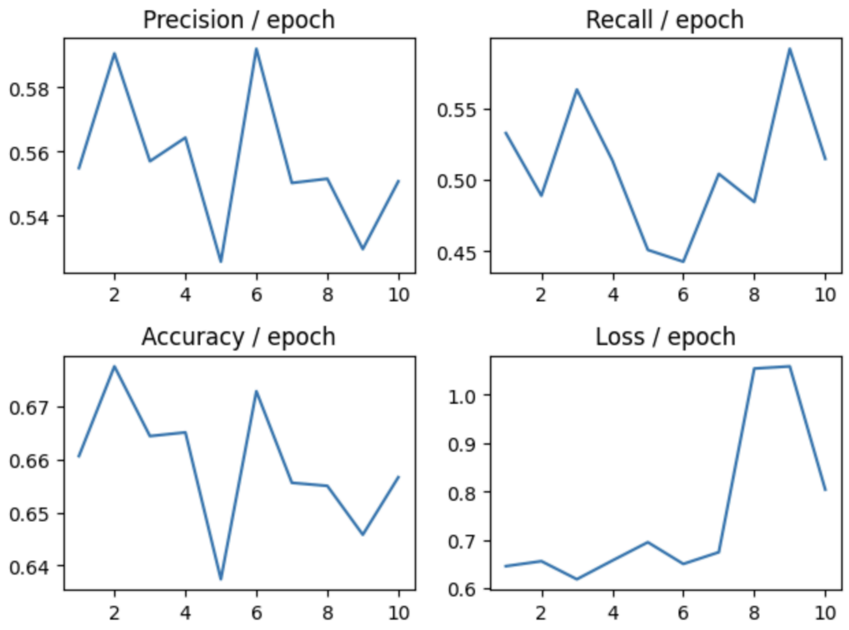
PureCNN_V3



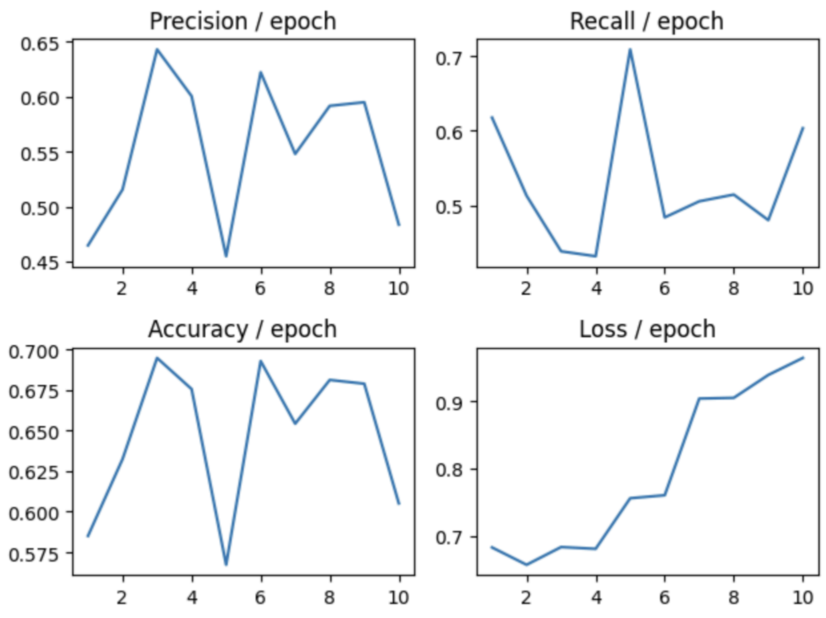
PureCNN_V4



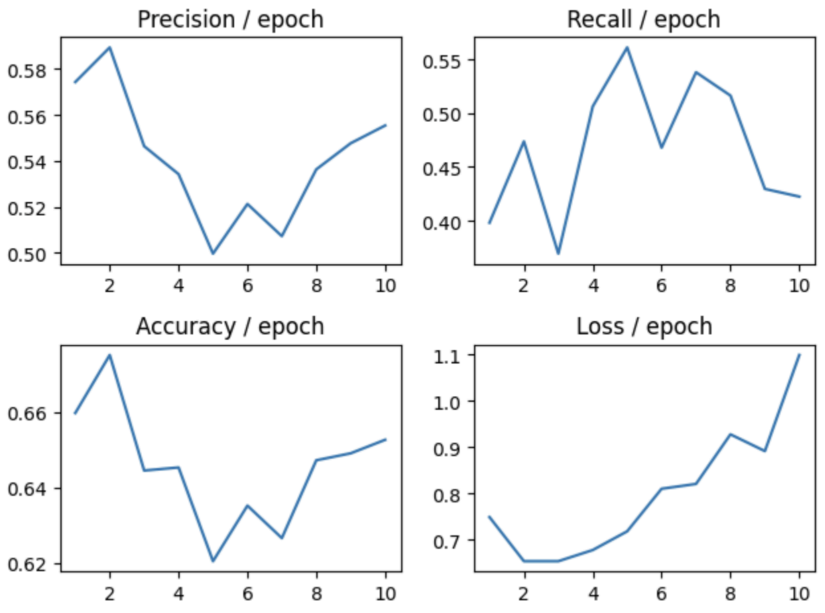
PureCNN_V5



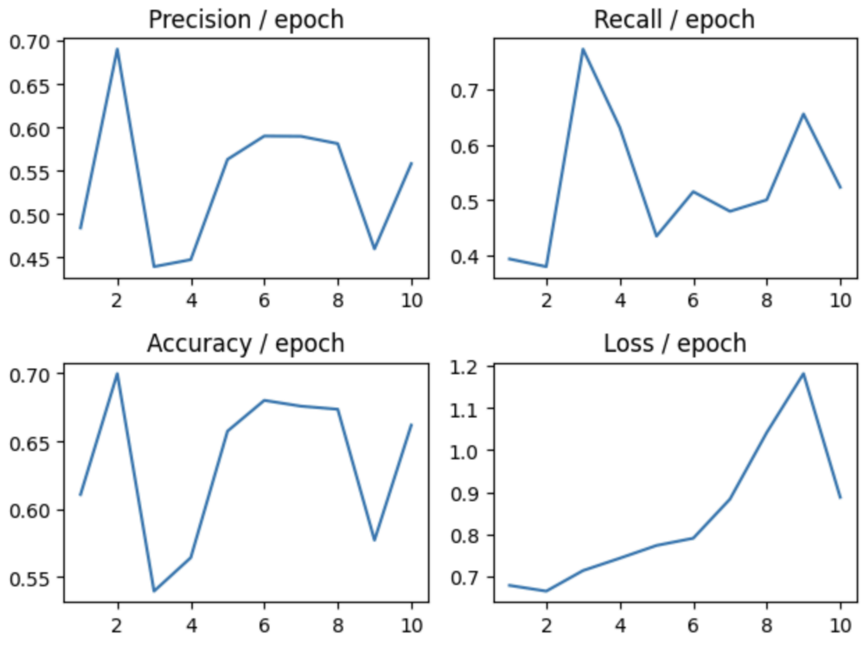
PureCNN_V6



PureCNN_V7

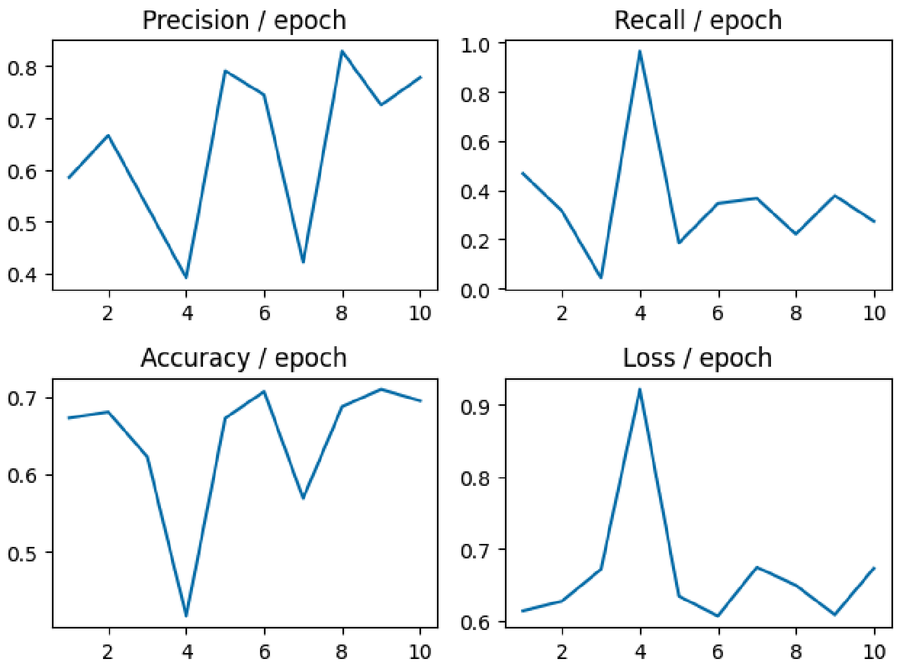


PureCNN_V8

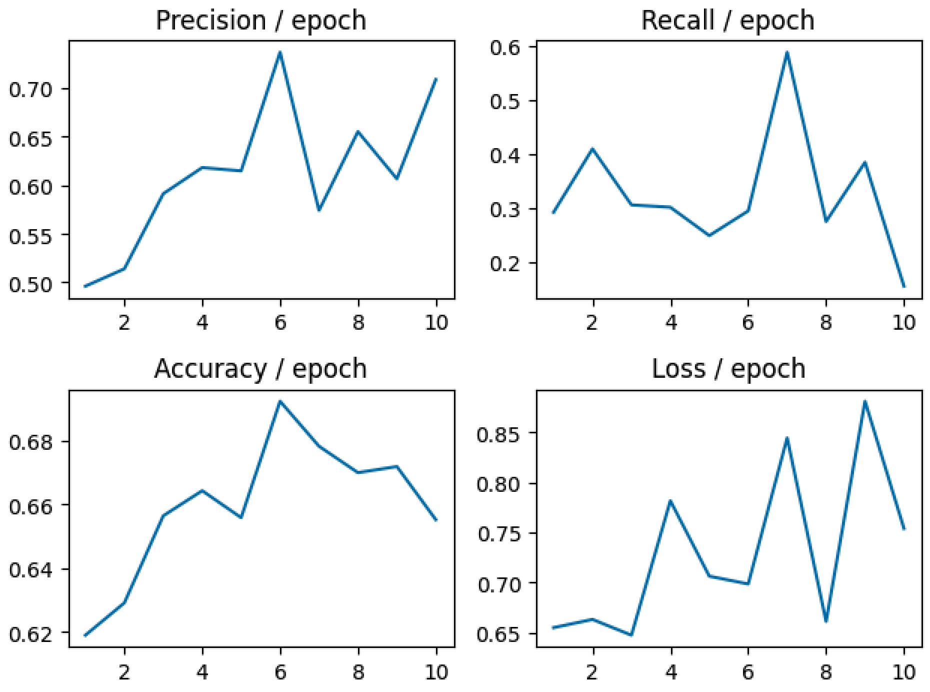


## CNN-LSTM:

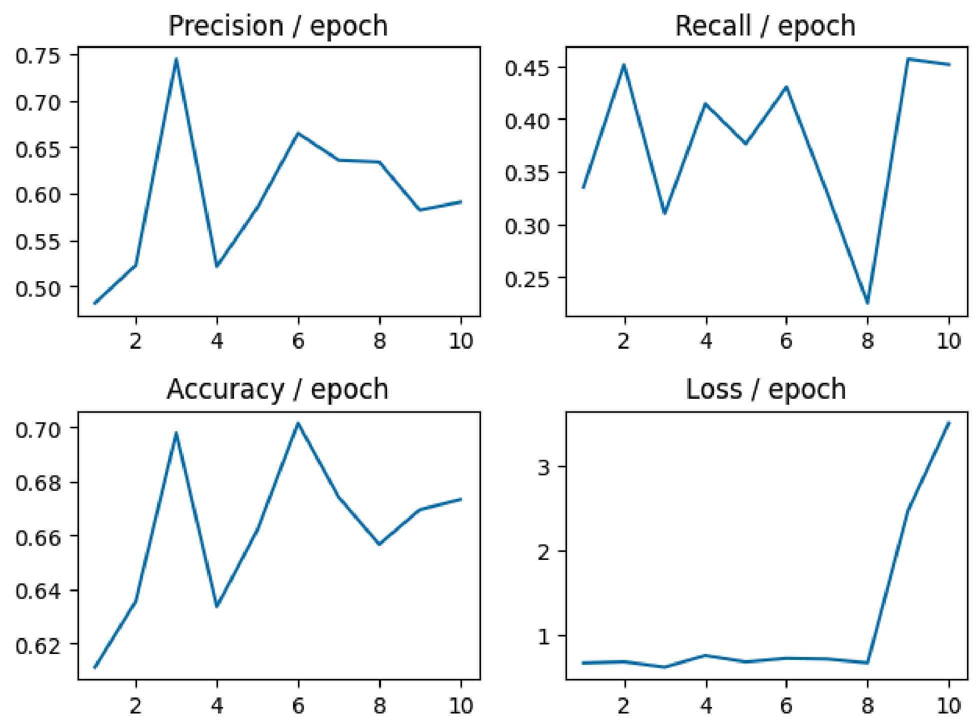
### CNN-LSTM V1



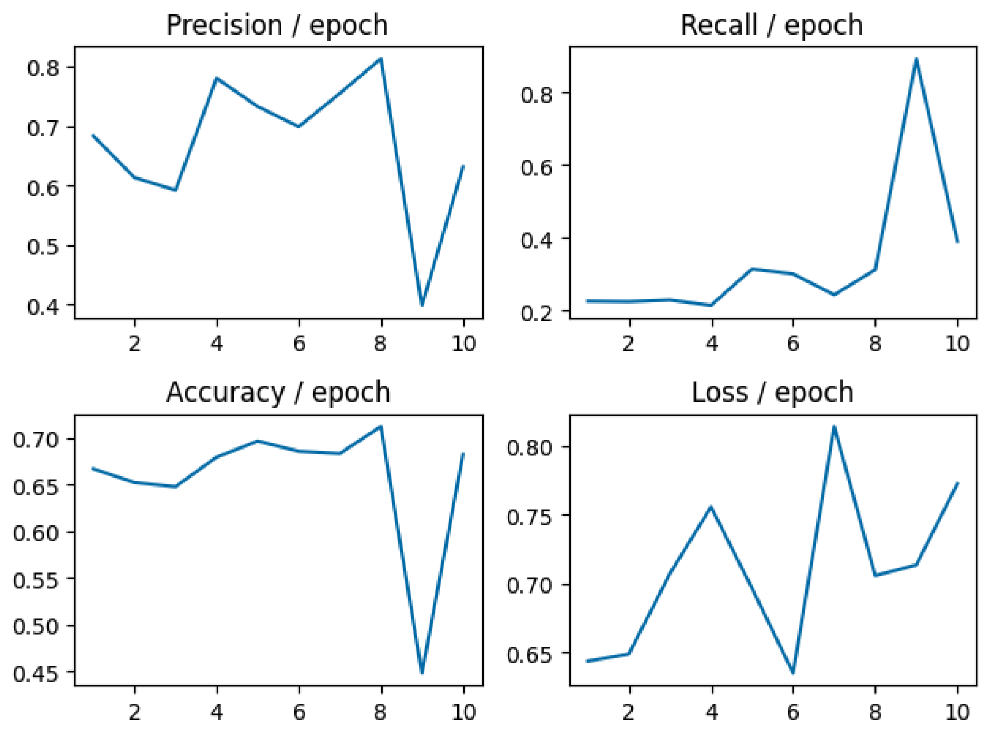
### CNN-LSTM V2



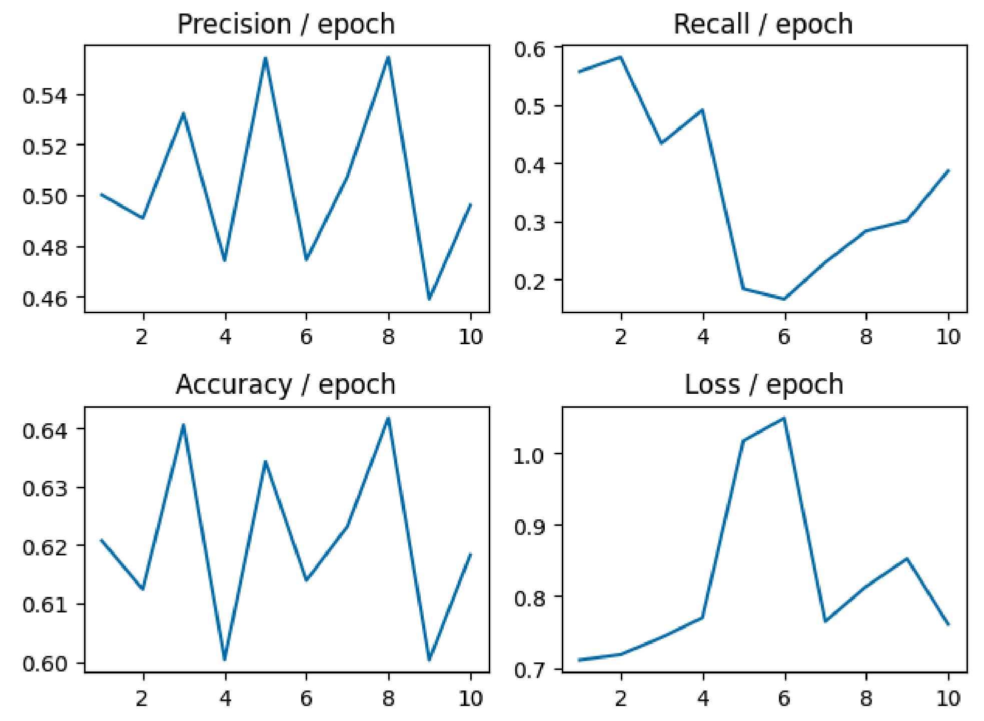
CNN-LSTM V3



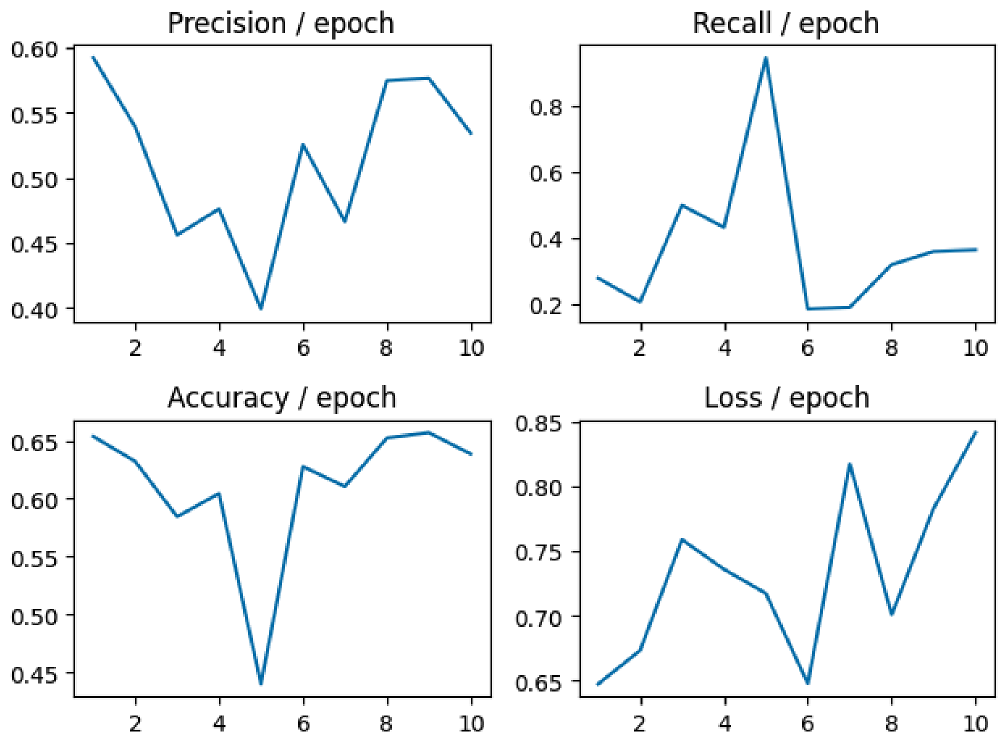
CNN-LSTM V4



## CNN-LSTM V5



## CNN LSTM V6


Precision / epoch
Recall / epoch
Accuracy / epoch
Loss / epoch