# Before you use this template

This template is just a recommended template for project Report. It only considers the general type of research in our paper pool. Feel free to edit it to better fit your project. You will iteratively update the same notebook submission for your draft and the final submission. Please check the project rubriks to get a sense of what is expected in the template.

# FAQ and Attentions

- Copy and move this template to your Google Drive. Name your notebook by your team ID (upper-left corner). Don't eidt this original file.
- This template covers most questions we want to ask about your reproduction experiment. You don't need to exactly follow the template, however, you should address the questions. Please feel free to customize your report accordingly.
- any report must have run-able codes and necessary annotations (in text and code comments).
- The notebook is like a demo and only uses small-size data (a subset of original data or processed data), the entire runtime of the notebook including data reading, data process, model training, printing, figure plotting, etc, must be within 8 min, otherwise, you may get penalty on the grade.
  - If the raw dataset is too large to be loaded you can select a subset of data and pre-process the data, then, upload the subset or processed data to Google Drive and load them in this notebook.
  - If the whole training is too long to run, you can only set the number of training epoch to a small number, e.g., 3, just show that the training is runable.
  - For results model validation, you can train the model outside this notebook in advance, then, load pretrained model and use it for validation (display the figures, print the metrics).
- The post-process is important! For post-process of the results,please use plots/figures. The code to summarize results and plot figures may be tedious, however, it won't be waste of time since these figures can be used for presentation. While plotting in code, the figures should have titles or captions if necessary (e.g., title your figure with "Figure 1. xxxx")

- There is not page limit to your notebook report, you can also use separate notebooks for the report, just make sure your grader can access and run/test them.
- If you use outside resources, please refer them (in any formats). Include the links to the resources if necessary.

## Mount Notebook to Google Drive

Upload the data, pretrianed model, figures, etc to your Google Drive, then mount this notebook to Google Drive. After that, you can access the resources freely.

Instruction: https://colab.research.google.com/notebooks/io.ipynb

Example: https://colab.research.google.com/drive/1srw_HFWQ2SMgmWIawucXfusGzrj1_U0q

Video: https://www.youtube.com/watch?v=zc8g8lGcwQU

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

## Introduction

Paper studied: Yo, Jang, Kwon, Lee, Jung, Byun, Jeong: ‘Predicting intraoperative hypotension using deep learning with waveforms of arterial blood pressure, electroencephalogram, and electrocardiogram: Retrospective study’ Plos One.
https://doi.org/10.1371/journal.pone.0272055 [1]

*Background*

Surgery is often associated with fluctuations in blood pressure.(Salmasi V, Maheshwari K, Yang D, Mascha EJ, Singh A, Sessler DI, et al. Relationship between intraoperative hypotension, defined by either reduction from baseline or absolute thresholds, and acute kidney and myocardial injury after noncardiac surgery: a retrospective cohort analysis. Anesthesiology. 2017;126(1):47–65. pmid:27792044 [2]) Low blood pressure (‘intraoperative hypotension’) might occur due to medications administered (such as sedation) or blood loss. Hypotension is potentially dangerous as it might lead to reduced blood flow to vital organs such as the heart or the brain. Therefore, careful monitoring of the intraoperative blood pressure is performed in order to treat hypotensive events (usually defined as a mean arterial blood pressure [MAP] <65 mmHg) if they occur (commonly with fluids or medications). Continuously measured parameters such as the MAP, the patient’s ECG, or EEG might allow for earlier prediction of subsequent hypotensive events. This could, in turn, enable a more timely intervention and potentially even prevention of hypotensive events.

*Paper explanation*

Specific approach: This paper uses a public data repository of vital signs taken during surgery in 10 operating rooms at Seoul National University Hospital between 01/06/2005 and 03/01/2024. The final analysis included 14,140 patients undergoing non-cardiac surgery. Arterial blood pressure (ABP), Electrocardiogram (ECG), and Electroencephalogram (EEG) waveforms obtained during surgery were used to predict hypotensive events. Specifically, 1-min intervals of the waveforms were sampled 3, 5, 10, and 15 min before a hypotensive event (defined as a MAP<65 mmHg ≥1 min) and compared to waveforms prior to ‘non-events’ (samples in the middle of a 30 min window of a MAP ≥75 mmHg). Unreliable cases were removed using the J signal quality index (Li Q., Mark R.G. & Clifford G.D. Artificial arterial blood pressure artifact models and an evaluation of a robust blood pressure and heart rate estimator. BioMed Eng OnLine. 2009; 8(13). pmid:19586547). Following data preprocessing, the authors trained a ResNet CNN for each waveform. The outputs are subsequently concatenated and passed through a classifier to predict hypotensive events.

## Scope of Reproducibility:

*Hypothesis:*

Is it possible to predict intraoperative hypotensive events using a deep-learning based analysis of MAP, ECG, and EEG waveforms?

*Experimental setup:*

Individual analysis of the predictive performance of the ABP, ECG, and EEG data compared to the model concatenating the results in order to understand the benefit vs the cost associated with the additional computation. Additional ablations will include studying the effect of the optimizer (Adam was used for the final model) and the learning rate (0.0001 in the paper).

# Methodology

Link to github repo: https://github.com/sebbeyer/DLH_project_168.git

Link to video presentation: https://mediaspace.illinois.edu/media/1_iclu5qwi

## Install and load packages

```
!pip install vitaldb
```

```
Collecting vitaldb
  Downloading vitaldb-1.4.8-py3-none-any.whl (56 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.8/56.8 kB 2.4 MB/s eta 0:00:
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pac
Collecting wfdb (from vitaldb)
  Downloading wfdb-4.1.2-py3-none-any.whl (159 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 160.0/160.0 kB 9.9 MB/s eta 0:0
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pytho
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
Requirement already satisfied: SoundFile>=0.10.0 in /usr/local/lib/python3.10
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.10
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-pa
Installing collected packages: wfdb, vitaldb
Successfully installed vitaldb-1.4.8 wfdb-4.1.2
```

```
!pip install scikit-plot
```

```
    Collecting scikit-plot
      Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
    Requirement already satisfied: matplotlib>=1.4.0 in /usr/local/lib/python3.10
    Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.1
    Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.10/dist-p
    Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.10/dist
    Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
    Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
    Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10
    Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10
    Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
    Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dis
    Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
    Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3
    Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pac
    Installing collected packages: scikit-plot
    Successfully installed scikit-plot-0.3.7
```

```
# import  packages you need
import os
import sys
import glob
import math
import copy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset
import vitaldb
import scipy.signal
import scipy.io.wavfile
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.metrics import average_precision_score, precision_recall_curve
from sklearn.metrics import auc
import scikitplot as skplt
import torch.nn.functional as F
```

```
# set seed
seed = 99
np.random.seed(seed)
torch.manual_seed(seed)
os.environ["PYTHONHASHSEED"] = str(seed)
```

## Load Data

*Source of the data:*

The dataset used is open access: https://osf.io/dtc45/. It can be obtained after signing the Data Use Agreement (https://vitaldb.net/docs/?documentId=1OyhiDYbN-VJ6TOme-Fkj4wbqJkVT3UazELcbCXcHmiY)

To run the notebook: 1) Download the .vital files after signing the Data Use Agreement (each .vital file corresponds to the tracings of one patient during surgery) 2) update 'raw_data_dir'

**The currently available dataset includes intraoperative recordings from 6,388 patients (https://vitaldb.net/dataset/). Interstingly, the original paper included 39,000 cases in the Vital DB. I am not sure whether the databse has been updated or whether the authors had access to additional cases. For this analysis, the currently available 6,388 patients will be included**

```
# show available tracks (all vital sign tracings obtained during surgery)
tracks = vitaldb.vital_trks('/content/drive/MyDrive/VitalDB/vital_files_1_250/000
tracks
```

```
def load_raw_data(raw_data_dir):

  # load data to lists with events and non-events (controls)
  # generate separate lists for 3, 5, 10, and 15 min preceding events

  ecg_3min = []
  art_3min = []
  eeg_3min = []
  y_3min = []

  ecg_5min = []
  art_5min = []
  eeg_5min = []
  y_5min = []
```

```
ecg_10min = []
art_10min = []
eeg_10min = []
y_10min = []

ecg_15min = []
art_15min = []
eeg_15min = []
y_15min = []

# tracks to load
tracks = ['SNUADC/ECG_II', 'SNUADC/ART', 'BIS/EEG1_WAV'] # the paper does not s
# the paper also does not specify which EEG waveform is being used -> Here, we

# iterate over all files
for file in os.listdir(raw_data_dir):

  #  extract ecg, blood pressure (art) and eeg data as numpy array
  vital_object = vitaldb.VitalFile(file, tracks)
  ecg_np = vital_object.to_numpy('SNUADC/ECG_II', 1/500)
  art_np = vital_object.to_numpy('SNUADC/ART', 1/500)
  eeg_np = vital_object.to_numpy('BIS/EEG1_WAV', 1/128)

  # calculate mean arterial blood pressure (MAP) for 1 minute intervals
  map = []

  for i in range(0, art_np.shape[0], 30000): # sampling of 500Hz -> 30,000 data
    map.append(np.mean(art_np[i:i+30000]))

  map = np.asarray(map)

  # Quality check - exclude implausible values
  map = np.where(map<20, np.nan, map)
  map = np.where(map>200, np.nan, map)

  # find index of events
  # hypotensive events: MAP<65 mmHg

  # index of hypotensive events mandating >20 min between each
  # hypotensive event (the paper does not specify whether this is
  # 20 min after the hypotension has completely resolved or 20 min
  # after the onset of hypotension)
  # -> here, I will require >20 min following the resolution of
  # hypotension
  # even more importantly, the authors did not specify how the following
  # scenario should be handled: a second hypotensive event w/in 20 min
  # of a prior hypotensive event with a third hypotensive event w/in 20 min
  # of the second hypotensive event but >20 min after the first hypotensive
  # event: should the third hypotensive event be considered an event and
  # included in the analysis???
  # -> here, I excluded even the third event as there are <20 min between
  # consecutive hypotensive events

  # the authors also didn't specify how they dealt with MAP>75 mmHg
  # segemnts lasting >30 (i.e. when during those longer interavls they
  # sampled controls)
  # -> here, I will use the initial 30 min of such intervals

  # 'events' array as indicator array:
    # '0' : MAP 65 - 75 mmHg
    # '1': MAP <65 mmHg with >20 min since the last hypotensive event
    # '-1': MAP <65 mmHg with <= 20 min since last hypotensive event
    # '2': MAP > 75 mmHg for 30 min (initial 30 min if MAP>75 for >30 min)

  events = (map<65)*1 # MAP<65
  events = np.where(np.isnan(map), np.nan, events) # keep nan as nan intervals

  prec_20_min = np.asarray([np.nansum(events[max(i[0]-20, 0):i[0]]) for i in er
  prec_20_min[prec_20_min >0] = 2

  events = events - prec_20_min
  events[events < -1] = 0

  map_75 = (map>75)*2 # MAP>75
  map_75 = np.where(np.isnan(map), np.nan, map_75) # keep nan as nan intervals

  prec_30_min = np.asarray([np.nansum(map_75[max(i[0]-30, 0):i[0]]) for i in er
  prec_30_min = (prec_30_min == 60).astype(int)

  prec_30_min_2 = np.asarray([np.nansum(prec_30_min[max(i[0]-30, 0):i[0]]) for
  prec_30_min_2[prec_30_min_2 >0] = 2

  map_75 = (map_75 * prec_30_min) - prec_30_min_2
  map_75[map_75 == -2] = 0

  events = events + map_75

  # append lists
```

```
  for i, j in enumerate(events):
    if j == 1:
      if (i >2 and np.isnan(ecg_np[(i-3)*30000:(i-2)*30000]).any() == False anc
      np.isnan(art_np[(i-3)*30000:(i-2)*30000]).any() == False and
      np.isnan(eeg_np[(i-3)*7680:(i-2)*7680]).any() == False and
      np.isnan(map[i-3]) == False):

        ecg_3min.append(ecg_np[(i-3)*30000:(i-2)*30000])
        art_3min.append(art_np[(i-3)*30000:(i-2)*30000])
        eeg_3min.append(eeg_np[(i-3)*7680:(i-2)*7680])
        y_3min.append(1)

      if (i >4 and np.isnan(ecg_np[(i-5)*30000:(i-4)*30000]).any() == False anc
      np.isnan(art_np[(i-5)*30000:(i-4)*30000]).any() == False and
      np.isnan(eeg_np[(i-5)*7680:(i-4)*7680]).any() == False and
      np.isnan(map[i-5]) == False):

        ecg_5min.append(ecg_np[(i-5)*30000:(i-4)*30000])
        art_5min.append(art_np[(i-5)*30000:(i-4)*30000])
        eeg_5min.append(eeg_np[(i-5)*7680:(i-4)*7680])
        y_5min.append(1)

      if (i >9 and np.isnan(ecg_np[(i-10)*30000:(i-9)*30000]).any() == False ar
      np.isnan(art_np[(i-10)*30000:(i-9)*30000]).any() == False and
      np.isnan(eeg_np[(i-10)*7680:(i-9)*7680]).any() == False and
      np.isnan(map[i-10]) == False):

        ecg_10min.append(ecg_np[(i-10)*30000:(i-9)*30000])
        art_10min.append(art_np[(i-10)*30000:(i-9)*30000])
        eeg_10min.append(eeg_np[(i-10)*7680:(i-9)*7680])
        y_10min.append(1)

      if (i >14 and np.isnan(ecg_np[(i-15)*30000:(i-14)*30000]).any() == False
      np.isnan(art_np[(i-15)*30000:(i-14)*30000]).any() == False and
      np.isnan(eeg_np[(i-15)*7680:(i-14)*7680]).any() == False and
      np.isnan(map[i-15]) == False):

        ecg_15min.append(ecg_np[(i-15)*30000:(i-14)*30000])
        art_15min.append(art_np[(i-15)*30000:(i-14)*30000])
        eeg_15min.append(eeg_np[(i-15)*7680:(i-14)*7680])
        y_15min.append(1)

    if (j == 2 and np.isnan(ecg_np[(i-15)*30000:(i-14)*30000]).any() == False a
    np.isnan(art_np[(i-15)*30000:(i-14)*30000]).any() == False and
```

```
      np.isnan(eeg_np[(i-15)*7680:(i-14)*7680]).any() == False and
      np.isnan(map[i-15]) == False):

        ecg_3min.append(ecg_np[(i-15)*30000:(i-14)*30000])
        ecg_5min.append(ecg_np[(i-15)*30000:(i-14)*30000])
        ecg_10min.append(ecg_np[(i-15)*30000:(i-14)*30000])
        ecg_15min.append(ecg_np[(i-15)*30000:(i-14)*30000])
        art_3min.append(art_np[(i-15)*30000:(i-14)*30000])
        art_5min.append(art_np[(i-15)*30000:(i-14)*30000])
        art_10min.append(art_np[(i-15)*30000:(i-14)*30000])
        art_15min.append(art_np[(i-15)*30000:(i-14)*30000])
        eeg_3min.append(eeg_np[(i-15)*7680:(i-14)*7680])
        eeg_5min.append(eeg_np[(i-15)*7680:(i-14)*7680])
        eeg_10min.append(eeg_np[(i-15)*7680:(i-14)*7680])
        eeg_15min.append(eeg_np[(i-15)*7680:(i-14)*7680])
        y_3min.append(0)
        y_5min.append(0)
        y_10min.append(0)
        y_15min.append(0)

  return (ecg_3min, art_3min, eeg_3min, y_3min, ecg_5min, art_5min, eeg_5min, y_5
         ecg_10min, art_10min, eeg_10min, y_10min, ecg_15min, art_15min, eeg_15mi



raw_data_dir = ['/content/drive/MyDrive/VitalDB/vital_files_1_6388']

ecg_3min = []
art_3min = []
eeg_3min = []
y_3min = []

ecg_5min = []
art_5min = []
eeg_5min = []
y_5min = []

ecg_10min = []
art_10min = []
eeg_10min = []
y_10min = []

ecg_15min = []
art_15min = []
eeg_15min = []
```

```
y_15min = []

for i in raw_data_dir:

  os.chdir(i)

  j = load_raw_data(i)
  ecg_3min.extend(j[0])
  art_3min.extend(j[1])
  eeg_3min.extend(j[2])
  y_3min.extend(j[3])

  ecg_5min.extend(j[4])
  art_5min.extend(j[5])
  eeg_5min.extend(j[6])
  y_5min.extend(j[7])

  ecg_10min.extend(j[8])
  art_10min.extend(j[9])
  eeg_10min.extend(j[10])
  y_10min.extend(j[11])

  ecg_15min.extend(j[12])
  art_15min.extend(j[13])
  eeg_15min.extend(j[14])
  y_15min.extend(j[15])
```

```
# convert to np array and save copy to google drive

ecg_3min = np.asarray(ecg_3min)
art_3min = np.asarray(art_3min)
eeg_3min = np.asarray(eeg_3min)
y_3min = np.asarray(y_3min)

ecg_5min = np.asarray(ecg_5min)
art_5min = np.asarray(art_5min)
eeg_5min = np.asarray(eeg_5min)
y_5min = np.asarray(y_5min)

ecg_10min = np.asarray(ecg_10min)
art_10min = np.asarray(art_10min)
eeg_10min = np.asarray(eeg_10min)
y_10min = np.asarray(y_10min)

ecg_15min = np.asarray(ecg_15min)
art_15min = np.asarray(art_15min)
eeg_15min = np.asarray(eeg_15min)
y_15min = np.asarray(y_15min)

os.chdir('/content/drive/MyDrive/VitalDB')

np.save('/content/drive/MyDrive/VitalDB/ecg_3min_6388.npy', ecg_3min)
np.save('/content/drive/MyDrive/VitalDB/art_3min_6388.npy', art_3min)
np.save('/content/drive/MyDrive/VitalDB/eeg_3min_6388.npy', eeg_3min)
np.save('/content/drive/MyDrive/VitalDB/y_3min_6388.npy', y_3min)

np.save('/content/drive/MyDrive/VitalDB/ecg_5min_6388.npy', ecg_5min)
np.save('/content/drive/MyDrive/VitalDB/art_5min_6388.npy', art_5min)
np.save('/content/drive/MyDrive/VitalDB/eeg_5min_6388.npy', eeg_5min)
np.save('/content/drive/MyDrive/VitalDB/y_5min_6388.npy', y_5min)

np.save('/content/drive/MyDrive/VitalDB/ecg_10min_6388.npy', ecg_10min)
np.save('/content/drive/MyDrive/VitalDB/art_10min_6388.npy', art_10min)
np.save('/content/drive/MyDrive/VitalDB/eeg_10min_6388.npy', eeg_10min)
np.save('/content/drive/MyDrive/VitalDB/y_10min_6388.npy', y_10min)

np.save('/content/drive/MyDrive/VitalDB/ecg_15min_6388.npy', ecg_15min)
np.save('/content/drive/MyDrive/VitalDB/art_15min_6388.npy', art_15min)
np.save('/content/drive/MyDrive/VitalDB/eeg_15min_6388.npy', eeg_15min)
np.save('/content/drive/MyDrive/VitalDB/y_15min_6388.npy', y_15min)
```

```
# load np arrays
ecg_3min = np.load('/content/drive/MyDrive/VitalDB/ecg_3min_6388.npy')
art_3min = np.load('/content/drive/MyDrive/VitalDB/art_3min_6388.npy')
eeg_3min = np.load('/content/drive/MyDrive/VitalDB/eeg_3min_6388.npy')
y_3min = np.load('/content/drive/MyDrive/VitalDB/y_3min_6388.npy')

ecg_5min = np.load('/content/drive/MyDrive/VitalDB/ecg_5min_6388.npy')
art_5min = np.load('/content/drive/MyDrive/VitalDB/art_5min_6388.npy')
eeg_5min = np.load('/content/drive/MyDrive/VitalDB/eeg_5min_6388.npy')
y_5min = np.load('/content/drive/MyDrive/VitalDB/y_5min_6388.npy')

ecg_10min = np.load('/content/drive/MyDrive/VitalDB/ecg_10min_6388.npy')
art_10min = np.load('/content/drive/MyDrive/VitalDB/art_10min_6388.npy')
eeg_10min = np.load('/content/drive/MyDrive/VitalDB/eeg_10min_6388.npy')
y_10min = np.load('/content/drive/MyDrive/VitalDB/y_10min_6388.npy')

ecg_15min = np.load('/content/drive/MyDrive/VitalDB/ecg_15min_6388.npy')
art_15min = np.load('/content/drive/MyDrive/VitalDB/art_15min_6388.npy')
eeg_15min = np.load('/content/drive/MyDrive/VitalDB/eeg_15min_6388.npy')
y_15min = np.load('/content/drive/MyDrive/VitalDB/y_15min_6388.npy')
```

## Preprocessing

```
# Apply frequency filter
# While the frequencies are provided in the paper, additional technical
# details such as the type of filter or the filter settings are not mentioned
# -> here, I am using a 4-th order Butterworth filter

def bandpass(data, edges, sampling_rate, poles: int = 4):
    sos = scipy.signal.butter(poles, edges, 'bandpass', fs=sampling_rate, output=
    filtered_data = scipy.signal.sosfiltfilt(sos, data, axis=1)
    return filtered_data
```

```
# Normalizing ECGs using Z scores
# It is not clear from the paper whether Z score are calculated for each sample
# or for the entire dataset

def normalize(data):

  mean_data = np.mean(data)
  sd_data = np.std(data)
  normalized_data = (data - mean_data) / sd_data
  return normalized_data
```

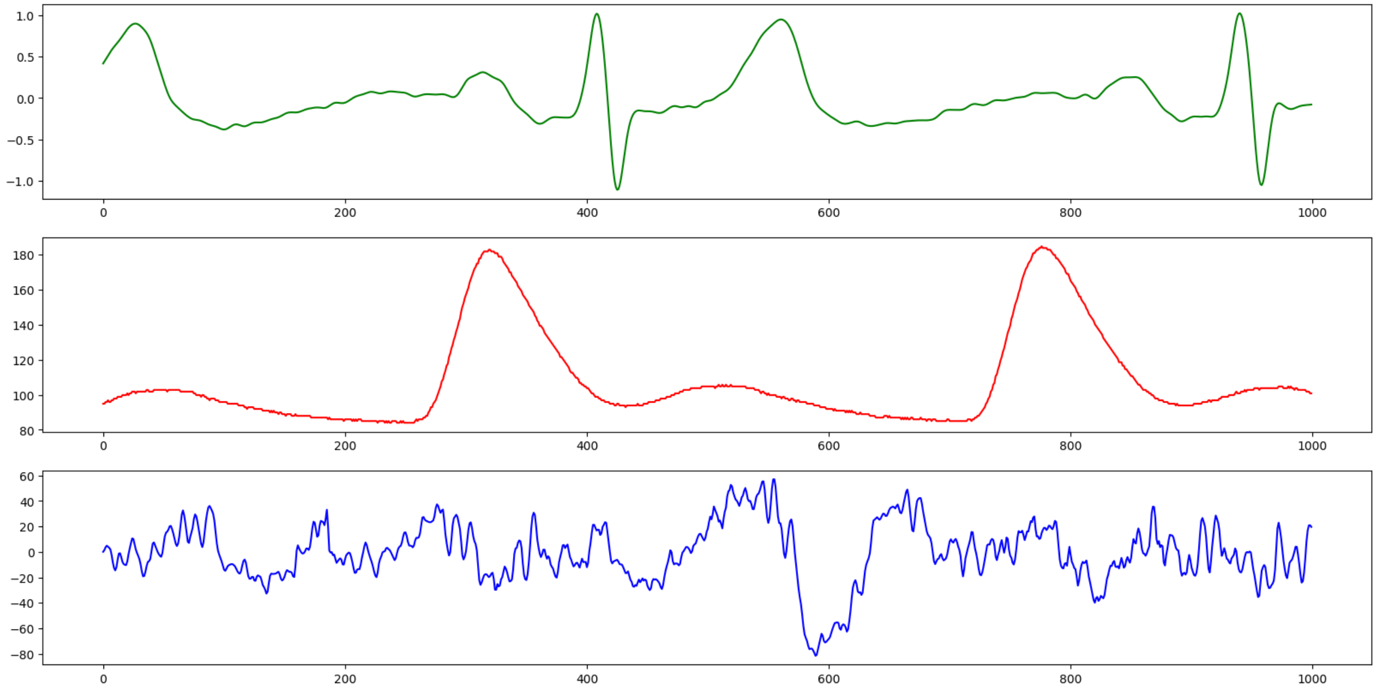
```
# process raw data
sampling_rate_ecg = 500
fmin_ecg = 1
fmax_ecg = 40

sampling_rate_eeg = 128
fmin_eeg = 0.5
fmax_eeg = 50

def process_data(ecg_3min, ecg_5min, ecg_10min, ecg_15min, eeg_3min, eeg_5min, ee
                 sampling_rate_ecg, fmin_ecg, fmax_ecg, sampling_rate_eeg, fmin_e

  # bandpass filter
  ecg_3min_filtered = bandpass(ecg_3min, [fmin_ecg, fmax_ecg], sampling_rate_ecg)
  ecg_5min_filtered = bandpass(ecg_5min, [fmin_ecg, fmax_ecg], sampling_rate_ecg)
  ecg_10min_filtered = bandpass(ecg_10min, [fmin_ecg, fmax_ecg], sampling_rate_ec
  ecg_15min_filtered = bandpass(ecg_15min, [fmin_ecg, fmax_ecg], sampling_rate_ec

  eeg_3min_filtered = bandpass(eeg_3min, [fmin_eeg, fmax_eeg], sampling_rate_eeg)
  eeg_5min_filtered = bandpass(eeg_5min, [fmin_eeg, fmax_eeg], sampling_rate_eeg)
  eeg_10min_filtered = bandpass(eeg_10min, [fmin_eeg, fmax_eeg], sampling_rate_ee
  eeg_15min_filtered = bandpass(eeg_15min, [fmin_eeg, fmax_eeg], sampling_rate_ee

  # normalizing ECG using Z score
  ecg_3min_normalized = normalize(ecg_3min_filtered)
  ecg_5min_normalized = normalize(ecg_5min_filtered)
  ecg_10min_normalized = normalize(ecg_10min_filtered)
  ecg_15min_normalized = normalize(ecg_15min_filtered)

  return ecg_3min_normalized, ecg_5min_normalized, ecg_10min_normalized, ecg_15mi
        eeg_3min_filtered, eeg_5min_filtered, eeg_10min_filtered, eeg_15min_filte

ecg_3min_normalized, ecg_5min_normalized, ecg_10min_normalized, ecg_15min_normali
eeg_3min_filtered, eeg_5min_filtered, eeg_10min_filtered, eeg_15min_filtered = \
process_data(ecg_3min, ecg_5min, ecg_10min, ecg_15min, eeg_3min, eeg_5min, eeg_10
                 sampling_rate_ecg, fmin_ecg, fmax_ecg, sampling_rate_eeg, fmin_e
```

## Statistics and Sample Tracings

```
# Statistics

# calculate statistics
def calculate_stats(y):

  n_samples_3min = len(y)
  cases_3min = np.sum(y)
  controls_3min = len(y) - np.sum(y)

  return n_samples_3min, cases_3min, controls_3min

n_samples_3min, cases_3min, controls_3min = calculate_stats(y_3min)

print(f'total number of samples with at least 3 minutes of data prior to event: {
print(f'total number of cases with at least 3 minutes of data prior to event: {ca
print(f'total number of controls: {controls_3min}')
```

```
    total number of samples with at least 3 minutes of data prior to event: 8903
    total number of cases with at least 3 minutes of data prior to event: 5078
    total number of controls: 3825
```

```
# Plot sample waveforms

ecg = ecg_3min_normalized[5500,:]
art = art_3min[1000,:]
eeg = eeg_3min_filtered[1000,:]

plt.figure(figsize=(20,10))
plt.subplot(311)
plt.plot(ecg[0:1000], color='g')
plt.subplot(312)
plt.plot(art[0:1000], color='r')
plt.subplot(313)
plt.plot(eeg[0:1000], color='b')
plt.show()
```



## Train / Val / Test Split and Dataset / Dataloader

```
# Define training, validation, and test samples (70:10:20 split)
# Since each patient might contribute multiple samples, shuffling will not be use

def split_train_val_test(ecg, art, eeg, y, cutoff_train_val, cutoff_val_test):

  split_train_val = int(cutoff_train_val * len(y))
  split_val_test = int(cutoff_val_test * len(y))

  ecg_train = ecg[:split_train_val]
  ecg_val = ecg[split_train_val:split_val_test]
  ecg_test = ecg[split_val_test:]

  art_train = art[:split_train_val]
  art_val = art[split_train_val:split_val_test]
  art_test = art[split_val_test:]

  eeg_train = eeg[:split_train_val]
  eeg_val = eeg[split_train_val:split_val_test]
  eeg_test = eeg[split_val_test:]

  y_train = y[:split_train_val]
  y_val = y[split_train_val:split_val_test]
  y_test = y[split_val_test:]

  return ecg_train, ecg_val, ecg_test, art_train, art_val, art_test, eeg_train, e

ecg_3min_train, ecg_3min_val, ecg_3min_test, art_3min_train, art_3min_val, art_3m
split_train_val_test(ecg_3min, art_3min, eeg_3min, y_3min, 0.7, 0.8)

ecg_5min_train, ecg_5min_val, ecg_5min_test, art_5min_train, art_5min_val, art_5m
split_train_val_test(ecg_5min_normalized, art_5min, eeg_5min_filtered, y_5min, 0.

ecg_10min_train, ecg_10min_val, ecg_10min_test, art_10min_train, art_10min_val, a
split_train_val_test(ecg_10min_normalized, art_10min, eeg_10min_filtered, y_10mir

ecg_15min_train, ecg_15min_val, ecg_15min_test, art_15min_train, art_15min_val, a
split_train_val_test(ecg_15min_normalized, art_15min, eeg_15min_filtered, y_15mir
# Define Custom Dataset class
from torch.utils.data import Dataset

class HypoDataset(Dataset):

    def __init__(self, ecg, art, eeg, y):

        super().__init__()
        self.y = y
        self.ecg = ecg
        self.art = art
        self.eeg = eeg

    def __len__(self):

        return len(self.y)

    def __getitem__(self, i):

        return ((self.ecg[i], self.art[i], self.eeg[i]), self.y[i])
```

```
# Define function to load dataset
from torch.utils.data import DataLoader

def load_data(dataset, batch_size=128):
    """
    Return a DataLoader instance basing on a Dataset instance, with batch_size sp
    """
    def my_collate(batch):

        # your code here
        x, y = zip(*batch)
        ecg, art, eeg = zip(*x)

        Y = torch.tensor(y, dtype=torch.float)

        ECG = torch.tensor(np.asarray(ecg), dtype=torch.float).transpose(1,2)
        ART = torch.tensor(np.asarray(art), dtype=torch.float).transpose(1,2)
        EEG = torch.tensor(np.asarray(eeg), dtype=torch.float).transpose(1,2)

        return (ECG, ART, EEG), Y

    return torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=Tr


train_loader_3min = load_data(HypoDataset(ecg_3min_train, art_3min_train, eeg_3mi
val_loader_3min = load_data(HypoDataset(ecg_3min_val, art_3min_val, eeg_3min_val,
test_loader_3min = load_data(HypoDataset(ecg_3min_test, art_3min_test, eeg_3min_t

train_loader_5min = load_data(HypoDataset(ecg_5min_train, art_5min_train, eeg_5mi
val_loader_5min = load_data(HypoDataset(ecg_5min_val, art_5min_val, eeg_5min_val,
test_loader_5min = load_data(HypoDataset(ecg_5min_test, art_5min_test, eeg_5min_t

train_loader_10min = load_data(HypoDataset(ecg_10min_train, art_10min_train, eeg_
val_loader_10min = load_data(HypoDataset(ecg_10min_val, art_10min_val, eeg_10min_
test_loader_10min = load_data(HypoDataset(ecg_10min_test, art_10min_test, eeg_10m

train_loader_15min = load_data(HypoDataset(ecg_15min_train, art_15min_train, eeg_
val_loader_15min = load_data(HypoDataset(ecg_15min_val, art_15min_val, eeg_15min_
test_loader_15min = load_data(HypoDataset(ecg_15min_test, art_15min_test, eeg_15m
```

## Model Architecture

```
    # the model architecture and details are poorly described and
    # discrepant. quite frankly, this makes me question the choice of
    # this paper as an option for a final project in this class:

    # 1)
    # the text mentions an additional encoder block (conv + dropout)
    # (before the data are passed through the residual blocks), which
    # is not shown in Fig 2 or Suppl. Table 1. Even more concerning, in the
    # text it says that the encoder blocks consist of a conv layer and a
    # max pooling layer whose technical specifications aren't mentioned
    # at all

    # 2)
    # at least some of the conv layers have to use padding since residual
    # connections are being used. However, padding is not mentioned at all

    # 3)
    # Supple Table 1 (detailing the hyperparameter settings) is not
    # consistent with the description of the model in the text of Fig 2:
    # the output size in Suppl Table 1 implies pooling layers are being
    # used between every other residual layer, but this is inconsistent
    # with the text or figure

    # 4)
    # According to Suppl. Table 1, channels increases w/ subsequent residual blocks,
    # but according to Fig 2 each layer has to conv layers - ???which layer
    # increases the channels???

    # 5)
    # According to Suppl. Table 1, kernel sizes change with subsequent
    # residual blocks - this is contradictory to what is mentioned in the text

    # 6)
    # What kind of activation function do the linear layers have? Relu???

    # 7)
    # Some residual blocks increase the number of channels -> a residual
    # connection is not possible here (unless the number of channels of the
    # input is also being adjusted for the skip connection, which is
    # also not mentioned at all in the text)

    # 8)
    # The output size numbers provided in Suppl Table 1 don't add up:
    # Max pooling w/ (2,2) applied to 1875 results in 937, not 938
    # Max pooling w/ (2,2) applied to 937 results in 468 (and even
    # max pooling applied to 938 does not result in 496)

    # 9)
    # it is unclear how weights were initialized

    # 10)
    # The authors do not mention the software / package they used for the analysis!

    class my_model(nn.Module):
      # use this class to define your model
      def __init__(self):

        super().__init__()

        self.encoder_ecg = nn.Sequential(
                            nn.Conv1d(in_channels=1, out_channels=1, kernel_size=3,st
                            )

        self.layer1_ecg = nn.Sequential(
                            nn.BatchNorm1d(1),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(1,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_ecg_layer1 = nn.MaxPool1d(2, 2)

        self.layer2_ecg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.layer3_ecg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_ecg_layer3 = nn.MaxPool1d(2, 2)

        self.layer4_ecg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.layer5_ecg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_ecg_layer5 = nn.MaxPool1d(2, 2)

        self.layer6_ecg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,4,15,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,15,stride=1, padding='same')
                            )

        self.layer7_ecg = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,7,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,7,stride=1, padding='same')
                            )

        self.maxpool_ecg_layer7 = nn.MaxPool1d(2, 2)

        self.layer8_ecg = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,7,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,7,stride=1, padding='same')
                            )

        self.layer9_ecg = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,7,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,7,stride=1, padding='same')
                            )

        self.maxpool_ecg_layer9 = nn.MaxPool1d(2, 2)

        self.layer10_ecg = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,6,7,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Conv1d(6,6,7,stride=1, padding='same')
                            )

        self.layer11_ecg = nn.Sequential(
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(6,6,7,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Conv1d(6,6,7,stride=1, padding='same')
                            )

        self.maxpool_ecg_layer11 = nn.MaxPool1d(2, 2)

        self.layer12_ecg = nn.Sequential(
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(6,6,7,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            )

        self.linear_ecg = nn.Linear(468*6, 32)

        self.encoder_art = nn.Sequential(
                            nn.Conv1d(1,1,3,stride=1, padding='same')
                            )

        self.layer1_art = nn.Sequential(
                            nn.BatchNorm1d(1),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(1,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_art_layer1 = nn.MaxPool1d(2, 2)

        self.layer2_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.layer3_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_art_layer3 = nn.MaxPool1d(2, 2)

        self.layer4_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.layer5_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_art_layer5 = nn.MaxPool1d(2, 2)

        self.layer6_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,4,15,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,15,stride=1, padding='same')
                            )

        self.layer7_art = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,7,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,7,stride=1, padding='same')
                            )

        self.maxpool_art_layer7 = nn.MaxPool1d(2, 2)

        self.layer8_art = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,7,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,7,stride=1, padding='same')
                            )

        self.layer9_art = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,7,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,7,stride=1, padding='same')
                            )

        self.maxpool_art_layer9 = nn.MaxPool1d(2, 2)

        self.layer10_art = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,6,7,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Conv1d(6,6,7,stride=1, padding='same')
                            )

        self.layer11_art = nn.Sequential(
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(6,6,7,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Conv1d(6,6,7,stride=1, padding='same')
                            )

        self.maxpool_art_layer11 = nn.MaxPool1d(2, 2)

        self.layer12_art = nn.Sequential(
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(6,6,7,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            )

        self.linear_art = nn.Linear(468*6, 32)

        self.encoder_eeg = nn.Sequential(
                            nn.Conv1d(1,1,3,stride=1, padding='same')
                            )

        self.layer1_eeg = nn.Sequential(
                            nn.BatchNorm1d(1),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(1,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer1 = nn.MaxPool1d(2, 2)

        self.layer2_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.layer3_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer3 = nn.MaxPool1d(2, 2)

        self.layer4_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.layer5_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer5 = nn.MaxPool1d(2, 2)

        self.layer6_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,4,7,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,7,stride=1, padding='same')
                            )

        self.layer7_eeg = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,3,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,3,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer7 = nn.MaxPool1d(2, 2)

        self.layer8_eeg = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,3,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,3,stride=1, padding='same')
                            )

        self.layer9_eeg = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,4,3,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Conv1d(4,4,3,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer9 = nn.MaxPool1d(2, 2)

        self.layer10_eeg = nn.Sequential(
                            nn.BatchNorm1d(4),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(4,6,3,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Conv1d(6,6,3,stride=1, padding='same')
                            )

        self.layer11_eeg = nn.Sequential(
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(6,6,3,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Conv1d(6,6,3,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer11 = nn.MaxPool1d(2, 2)

        self.layer12_eeg = nn.Sequential(
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(6,6,3,stride=1, padding='same'),
                            nn.BatchNorm1d(6),
                            nn.ReLU(),
                            )

        self.linear_eeg = nn.Linear(120*6, 32)

        self.linear_combined1 = nn.Linear(96, 16)
        self.linear_combined2 = nn.Linear(16, 1)

        self.apply(self._init_weights)

      def _init_weights(self, module):

          if isinstance(module, nn.Conv1d):
            nn.init.kaiming_normal_(module.weight, nonlinearity='relu')

            if module.bias is not None:
              nn.init.zeros_(module.bias)

          elif isinstance(module, nn.BatchNorm1d):
            nn.init.constant_(module.weight, 1)
            nn.init.constant_(module.bias, 0)

          elif isinstance(module, nn.Linear):
            nn.init.xavier_normal_(module.weight)

            if module.bias is not None:
              nn.init.zeros_(module.bias)

      def forward(self, ecg, art, eeg):
        ecg = self.encoder_ecg(ecg)
        tmp = self.layer1_ecg(ecg)
        ecg = tmp + ecg
        ecg = self.maxpool_ecg_layer1(ecg)
        tmp = self.layer2_ecg(ecg)
        ecg = tmp + ecg
        tmp = self.layer3_ecg(ecg)
        ecg = tmp + ecg
        ecg = self.maxpool_ecg_layer3(ecg)
        tmp = self.layer4_ecg(ecg)
        ecg = tmp + ecg
        tmp = self.layer5_ecg(ecg)
        ecg = tmp + ecg
        ecg = self.maxpool_ecg_layer5(ecg)
        ecg = self.layer6_ecg(ecg)
        tmp = self.layer7_ecg(ecg)
        ecg = tmp + ecg
        ecg = self.maxpool_ecg_layer7(ecg)
        tmp = self.layer8_ecg(ecg)
        ecg = tmp + ecg
        tmp = self.layer9_ecg(ecg)
        ecg = tmp + ecg
        ecg = self.maxpool_ecg_layer9(ecg)
        ecg = self.layer10_ecg(ecg)
        tmp = self.layer11_ecg(ecg)
        ecg = tmp + ecg
        ecg = self.maxpool_ecg_layer11(ecg)
        tmp = self.layer12_ecg(ecg)
        ecg = tmp + ecg
        ecg = torch.flatten(ecg, 1)
        ecg = F.relu(self.linear_ecg(ecg))

        art = self.encoder_art(art)
        tmp = self.layer1_art(art)
        art = tmp + art
        art = self.maxpool_art_layer1(art)
        tmp = self.layer2_art(art)
        art = tmp + art
        tmp = self.layer3_art(art)
        art = tmp + art
        art = self.maxpool_art_layer3(art)
        tmp = self.layer4_art(art)
        art = tmp + art
        tmp = self.layer5_art(art)
        art = tmp + art
        art = self.maxpool_art_layer5(art)
        art = self.layer6_art(art)
        tmp = self.layer7_art(art)
        art = tmp + art
        art = self.maxpool_art_layer7(art)
        tmp = self.layer8_art(art)
        art = tmp + art
        tmp = self.layer9_art(art)
        art = tmp + art
        art = self.maxpool_art_layer9(art)
        art = self.layer10_art(art)
        tmp = self.layer11_art(art)
        art = tmp + art
        art = self.maxpool_art_layer11(art)
        tmp = self.layer12_art(art)
        art = tmp + art
        art = torch.flatten(art, 1)
        art = F.relu(self.linear_art(art))

        eeg = self.encoder_eeg(eeg)
        tmp = self.layer1_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer1(eeg)
        tmp = self.layer2_eeg(eeg)
        eeg = tmp + eeg
        tmp = self.layer3_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer3(eeg)
        tmp = self.layer4_eeg(eeg)
        eeg = tmp + eeg
        tmp = self.layer5_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer5(eeg)
        eeg = self.layer6_eeg(eeg)
        tmp = self.layer7_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer7(eeg)
        tmp = self.layer8_eeg(eeg)
        eeg = tmp + eeg
        tmp = self.layer9_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer9(eeg)
        eeg = self.layer10_eeg(eeg)
        tmp = self.layer11_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer11(eeg)
        tmp = self.layer12_eeg(eeg)
        eeg = tmp + eeg
        eeg = torch.flatten(eeg, 1)
        eeg = F.relu(self.linear_eeg(eeg))

        combined = F.relu(self.linear_combined1(torch.cat((ecg, art, eeg), -1)))
        logits = self.linear_combined2(combined)

        return logits
```

## Training

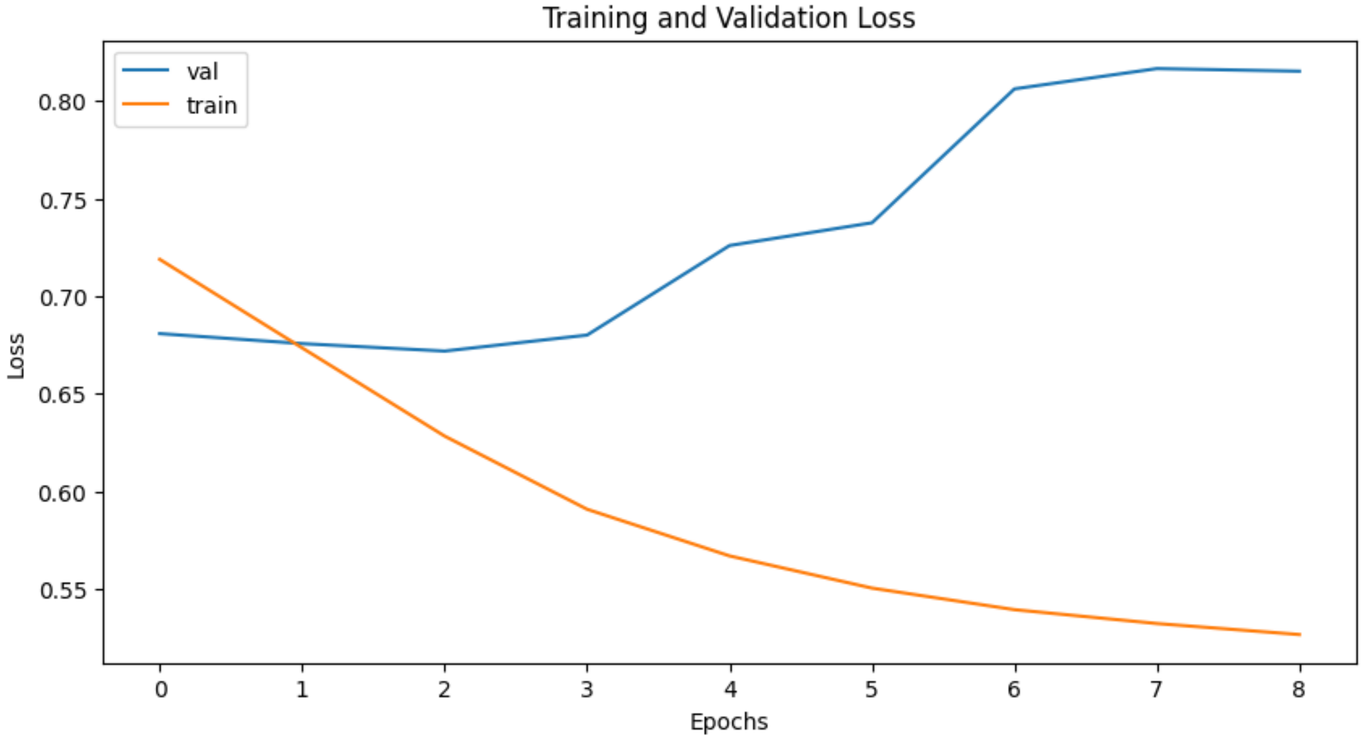
```
# define function to train for one epoch and return the model state,
# epoch training loss, and epoch training accuracy

def train_model_one_iter(train_dataloader, model, loss_func, optimizer):

  model.train()
  running_loss = 0
  total_correct = 0
  total_samples = 0

  for (ecg, art, eeg), y in train_dataloader:

    logits = model(ecg, art, eeg)
    loss = loss_func(logits.view(logits.shape[0]), y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    running_loss += loss.item()*y.size(0)

    # calculate accuracy for current batch
    y_hat = torch.sigmoid(model(ecg, art, eeg))
    y_hat = y_hat.detach()
    y_hat = (y_hat>0.5).int()
    total_correct += torch.sum(y_hat.view(y_hat.shape[0]) == y)
    total_samples += y.size(0)

    # print(y_hat.size())
    # print(y_hat)
    # print(total_samples)
    # print(y)

  epoch_loss = running_loss / len(train_dataloader.dataset)
  epoch_accuracy = total_correct / total_samples

  return model, optimizer, epoch_loss, epoch_accuracy
```

```
# define function to calculate validation loss and return
# validation loss and validation epoch accuracy

def calc_val_loss(val_dataloader, model, loss_func):

  model.eval()
  valid_loss = 0
  total_correct = 0
  total_samples = 0

  with torch.no_grad():
    for batch_idx, ((ecg, art, eeg), y) in enumerate(val_dataloader):

      logits = model(ecg, art, eeg)
      loss = loss_func(logits.view(logits.shape[0]), y)

      valid_loss += (
          (1 / (batch_idx + 1)) * (loss.data.item() - valid_loss)
        )
      # val_running_loss += loss.item() * y.size(0)

      # calculate accuracy for current batch
      y_hat = torch.sigmoid(model(ecg, art, eeg))
      y_hat = y_hat.detach()
      y_hat = (y_hat>0.5).int()
      total_correct += (y_hat.view(y_hat.shape[0]) == y).sum().item()
      total_samples += y.size(0)

  # val_epoch_loss = val_running_loss / len(val_dataloader.dataset)
  epoch_acc = total_correct / total_samples

  return valid_loss, epoch_acc
```

```
# define functions to save and load model checkpoints

def checkpoint(model, optimizer, filename):
    torch.save({
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            }, filename)

def resume(model, optimizer, filename):

  checkpoint = torch.load(filename)
  model.load_state_dict(checkpoint['model_state_dict'])
  optimizer.load_state_dict(checkpoint['optimizer_state_dict'])



# Train model

def model_train(train_loader, val_loader, model, loss_func, optimizer, num_epoch,

  train_losses = []
  val_losses = []
  train_acc = []
  val_acc = []

  best_val_loss = 9999999
  best_epoch = -1

  for i in range(num_epoch):

    model, optimizer, train_epoch_loss, train_epoch_acc = train_model_one_iter(tr
    print(f'Epoch: {i}, Train loss: {train_epoch_loss}, Train accuracy: {train_ep
    train_losses.append(train_epoch_loss)
    train_acc.append(train_epoch_acc)


    val_epoch_loss, val_epoch_acc = calc_val_loss(val_loader, model, loss_func)
    print(f'Val loss: {val_epoch_loss}, Val accuracy: {val_epoch_acc}')
    val_losses.append(val_epoch_loss)
    val_acc.append(val_epoch_acc)

    if val_epoch_loss <= best_val_loss:
        best_val_loss = val_epoch_loss
        best_epoch = i
        checkpoint(model, optimizer, best_model_path)
    elif (i - best_epoch) > early_stop_thresh:
        print(f'Early stopped training at epoch {i}')
        break  # terminate the training loop

    # if es.step(val_epoch_loss):
    #   break

  # show train and val losses
  plt.figure(figsize=(10,5))
  plt.title("Training and Validation Loss")
  plt.plot(val_losses,label="val")
  plt.plot(train_losses,label="train")
  plt.xlabel("Epochs")
  plt.ylabel("Loss")
  plt.legend()
  plt.show()

  return model
```

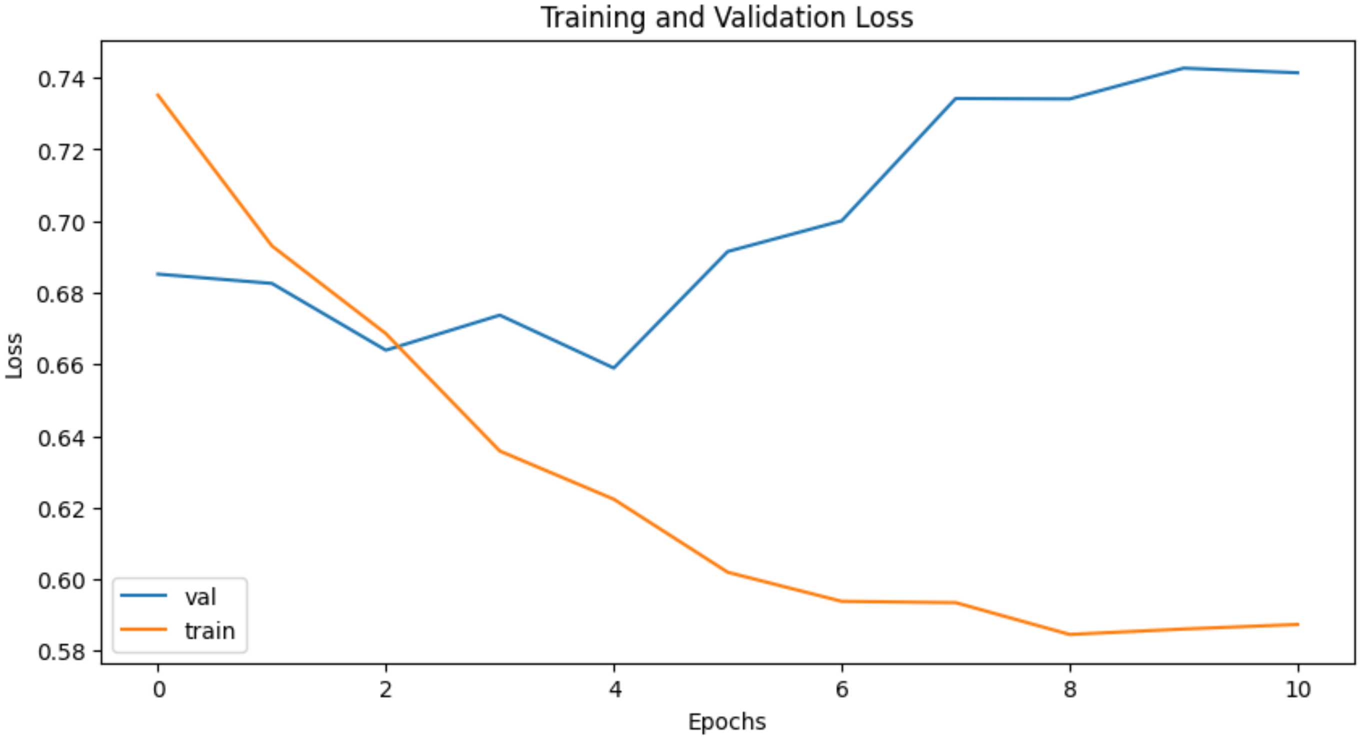
## Replicate paper: Train models for 3, 5, 10, and 15 min with Adam, lr=0.0001, and patience=5

```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min.pth"
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.7188880445868427, Train accuracy: 0.5505455732345581
Val loss: 0.6808600766318185, Val accuracy: 0.5853932584269663
Epoch: 1, Train loss: 0.673528744565354, Train accuracy: 0.6083119511604309
Val loss: 0.6757329446928841, Val accuracy: 0.6101123595505618
Epoch: 2, Train loss: 0.628377544252496, Train accuracy: 0.6497111916542053
Val loss: 0.6719142198562622, Val accuracy: 0.6606741573033708
Epoch: 3, Train loss: 0.5909076030049306, Train accuracy: 0.6903080940246582
Val loss: 0.6801055244037083, Val accuracy: 0.5853932584269663
Epoch: 4, Train loss: 0.5670231028300959, Train accuracy: 0.7069961428642273
Val loss: 0.7259670751435416, Val accuracy: 0.4280898876404494
Epoch: 5, Train loss: 0.5505311880858474, Train accuracy: 0.7171052694320679
Val loss: 0.7376130138124738, Val accuracy: 0.4258426966292135
Epoch: 6, Train loss: 0.5394882662305477, Train accuracy: 0.7259306907653809
Val loss: 0.8061788763318744, Val accuracy: 0.4157303370786517
Epoch: 7, Train loss: 0.5323938742299135, Train accuracy: 0.7321887016296387
Val loss: 0.816577638898577, Val accuracy: 0.41685393258426967
Epoch: 8, Train loss: 0.5267526160002673, Train accuracy: 0.7294608354568481
Val loss: 0.8152453218187604, Val accuracy: 0.41797752808988764
Early stopped training at epoch 8
```

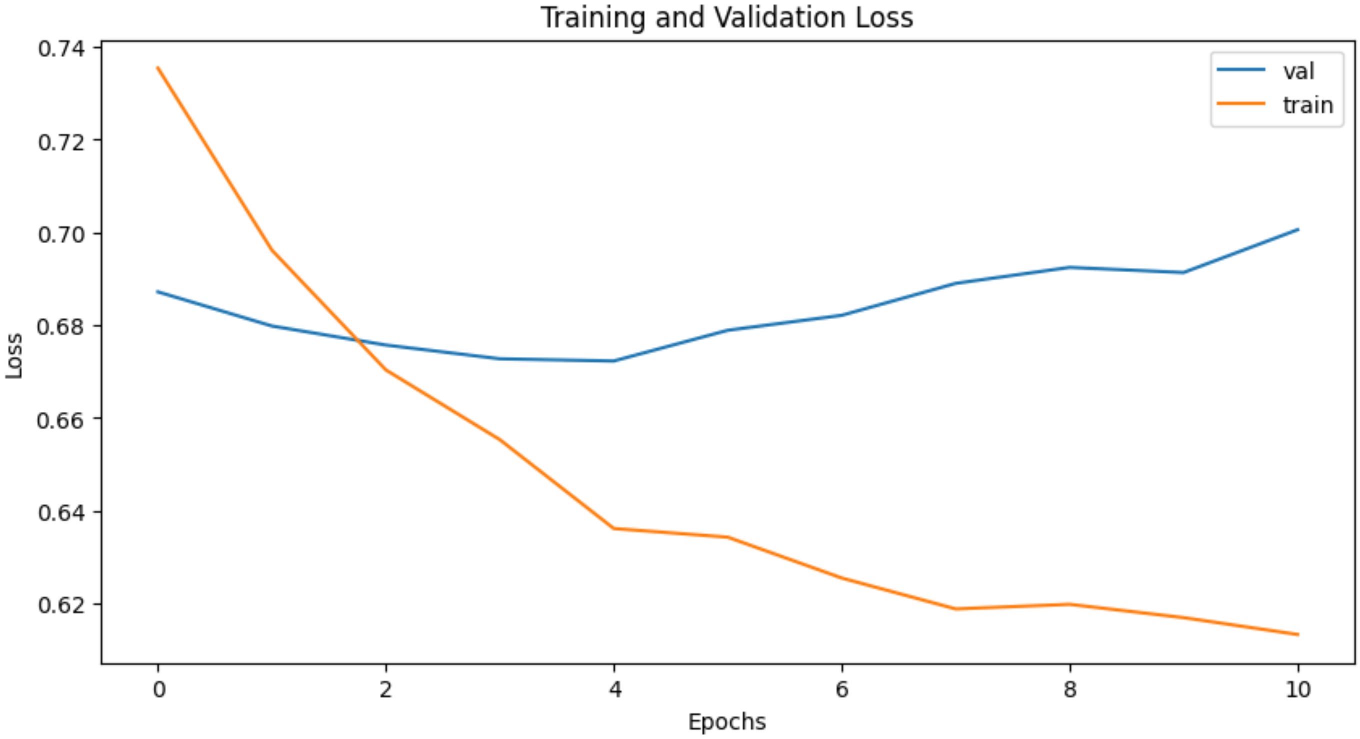


```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
```

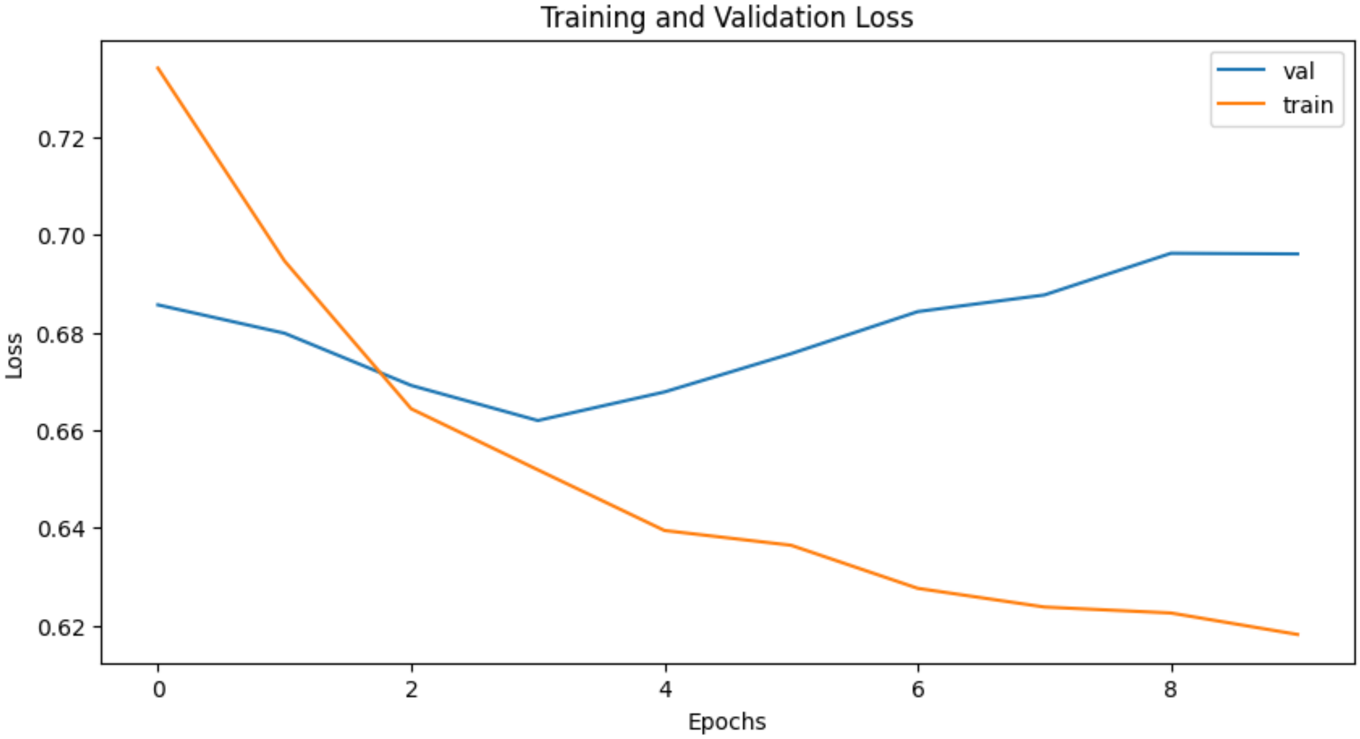
```
num_epoch = 100
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.7351230914687186, Train accuracy: 0.5422797203063965
Val loss: 0.6851714849472046, Val accuracy: 0.5769230769230769
Epoch: 1, Train loss: 0.693054606746837, Train accuracy: 0.5671786665916443
Val loss: 0.682573812348502, Val accuracy: 0.5859728506787331
Epoch: 2, Train loss: 0.6685662267568522, Train accuracy: 0.6003233790397644
Val loss: 0.6639316933495658, Val accuracy: 0.6142533936651584
Epoch: 3, Train loss: 0.6357869825733219, Train accuracy: 0.6339530944824219
Val loss: 0.67371324130467, Val accuracy: 0.6414027149321267
Epoch: 4, Train loss: 0.6223419961663341, Train accuracy: 0.6575586199760437
Val loss: 0.658958579812731, Val accuracy: 0.6493212669683258
Epoch: 5, Train loss: 0.6019495085351585, Train accuracy: 0.6649959683418274
Val loss: 0.6914824502808706, Val accuracy: 0.5599547511312217
Epoch: 6, Train loss: 0.5938345197814072, Train accuracy: 0.6729183793067932
Val loss: 0.7000499878610884, Val accuracy: 0.5248868778280543
Epoch: 7, Train loss: 0.5934698602896885, Train accuracy: 0.6748585104942322
Val loss: 0.7341888887541634, Val accuracy: 0.41402714932126694
Epoch: 8, Train loss: 0.5845873979762762, Train accuracy: 0.677930474281311
Val loss: 0.7340602874755859, Val accuracy: 0.415158371040724
Epoch: 9, Train loss: 0.5861077360646225, Train accuracy: 0.6738884449005127
Val loss: 0.7426630514008657, Val accuracy: 0.416289592760181
Epoch: 10, Train loss: 0.5873817382267433, Train accuracy: 0.6746968626976013
Val loss: 0.7413639596530369, Val accuracy: 0.41402714932126694
Early stopped training at epoch 10
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5

# 10 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min.pth"
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
Epoch: 0, Train loss: 0.735423487169121, Train accuracy: 0.539933979511261
Val loss: 0.6871904305049351, Val accuracy: 0.5704387990762124
Epoch: 1, Train loss: 0.6961830202895816, Train accuracy: 0.5676567554473877
Val loss: 0.6798341018812997, Val accuracy: 0.5681293302540416
Epoch: 2, Train loss: 0.6703617642814964, Train accuracy: 0.5968647003173828
Val loss: 0.67572420835495, Val accuracy: 0.5946882217090069
Epoch: 3, Train loss: 0.6553234470952856, Train accuracy: 0.6087458729743958
Val loss: 0.6727611422538757, Val accuracy: 0.6235565819861432
Epoch: 4, Train loss: 0.6361695448164105, Train accuracy: 0.6163366436958313
Val loss: 0.6722898823874337, Val accuracy: 0.6316397228637414
Epoch: 5, Train loss: 0.634307070337113, Train accuracy: 0.6245874762535095
Val loss: 0.6789095742361886, Val accuracy: 0.5935334872979214
Epoch: 6, Train loss: 0.6255021655126767, Train accuracy: 0.6268976926803589
Val loss: 0.6821337001664298, Val accuracy: 0.5635103926096998
Epoch: 7, Train loss: 0.6188389621158638, Train accuracy: 0.6356435418128967
Val loss: 0.6890221067837307, Val accuracy: 0.5415704387990762
Epoch: 8, Train loss: 0.6198284913997839, Train accuracy: 0.6429042816162109
Val loss: 0.692468558038984, Val accuracy: 0.5288683602771362
Epoch: 9, Train loss: 0.6169599273810685, Train accuracy: 0.6359735727310181
Val loss: 0.6913609249251229, Val accuracy: 0.5346420323325635
Epoch: 10, Train loss: 0.6133306949052086, Train accuracy: 0.6363036036491394
Val loss: 0.7005865403584072, Val accuracy: 0.4341801385681293
Early stopped training at epoch 10
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min.pth"
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
Epoch: 0, Train loss: 0.7341364590316345, Train accuracy: 0.5423474311828613
Val loss: 0.6856723683221, Val accuracy: 0.5682613768961493
Epoch: 1, Train loss: 0.6946537988747943, Train accuracy: 0.573524534702301
Val loss: 0.6798610091209412, Val accuracy: 0.5682613768961493
Epoch: 2, Train loss: 0.6643923624987282, Train accuracy: 0.6098699569702148
Val loss: 0.6691758292061942, Val accuracy: 0.5927654609101517
Epoch: 3, Train loss: 0.6518971863370134, Train accuracy: 0.6265421509742737
Val loss: 0.661989552634103, Val accuracy: 0.6149358226371062
Epoch: 4, Train loss: 0.6394752939767383, Train accuracy: 0.6267089247703552
Val loss: 0.6678378241402761, Val accuracy: 0.6336056009334889
Epoch: 5, Train loss: 0.6364510892311228, Train accuracy: 0.6357119083404541
Val loss: 0.6756750004632132, Val accuracy: 0.617269544924154
Epoch: 6, Train loss: 0.6276340020144133, Train accuracy: 0.6342114210128784
Val loss: 0.6842840228761946, Val accuracy: 0.5484247374562428
Epoch: 7, Train loss: 0.6238071990831966, Train accuracy: 0.6365455389022827
Val loss: 0.6876937236104693, Val accuracy: 0.5379229871645275
Epoch: 8, Train loss: 0.6225820369384971, Train accuracy: 0.6453818082809448
Val loss: 0.6962312715394156, Val accuracy: 0.49474912485414235
Epoch: 9, Train loss: 0.6181862294654045, Train accuracy: 0.6375458240509033
Val loss: 0.6960996389389038, Val accuracy: 0.48891481913652274
Early stopped training at epoch 9
```

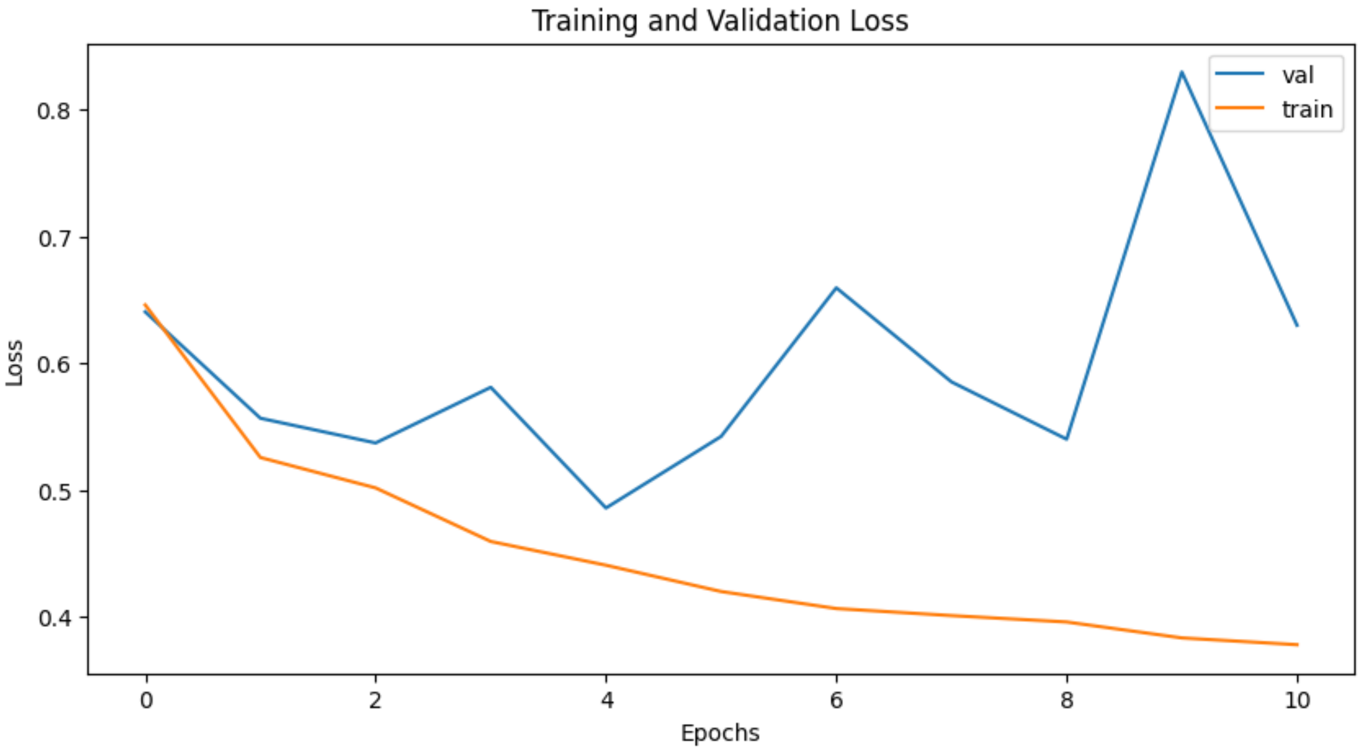


## Hyperparameter Search

Test different batch sizes (128, 64, and 32) and learning rates (0.0001 and 0.001)

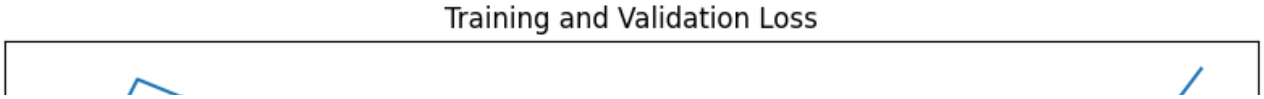
### batch size 128, learning rate of 0.001

```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_adam_lr001.pth"
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.6460492081069824, Train accuracy: 0.6463414430618286
Val loss: 0.6405951806477138, Val accuracy: 0.6696629213483146
Epoch: 1, Train loss: 0.5257856170234386, Train accuracy: 0.7350770235061646
Val loss: 0.5567381722586495, Val accuracy: 0.7202247191011236
Epoch: 2, Train loss: 0.5018375786033307, Train accuracy: 0.7479140162467957
Val loss: 0.5372666205678668, Val accuracy: 0.7247191011235955
Epoch: 3, Train loss: 0.4595930694156495, Train accuracy: 0.7793645858764648
Val loss: 0.5811289804322379, Val accuracy: 0.7314606741573034
Epoch: 4, Train loss: 0.44088754532272917, Train accuracy: 0.7875481247901917
Val loss: 0.485974086182458, Val accuracy: 0.749438202247191
Epoch: 5, Train loss: 0.42014824219325386, Train accuracy: 0.8013479113578796
Val loss: 0.5424096243722099, Val accuracy: 0.7370786516853932
Epoch: 6, Train loss: 0.40671773914807263, Train accuracy: 0.8084082007408142
Val loss: 0.6595593435423714, Val accuracy: 0.7235955056179775
Epoch: 7, Train loss: 0.40116178576233147, Train accuracy: 0.8084082007408142
Val loss: 0.5854069505419051, Val accuracy: 0.7224719101123596
Epoch: 8, Train loss: 0.3961289358613427, Train accuracy: 0.8170731663703918
Val loss: 0.540217821087156, Val accuracy: 0.7359550561797753
Epoch: 9, Train loss: 0.3835608602258758, Train accuracy: 0.8209242820739746
Val loss: 0.8296746781894139, Val accuracy: 0.6786516853932584
Epoch: 10, Train loss: 0.3782253468908915, Train accuracy: 0.8315147757530212
Val loss: 0.6300536564418248, Val accuracy: 0.6831460674157304
Early stopped training at epoch 10
```

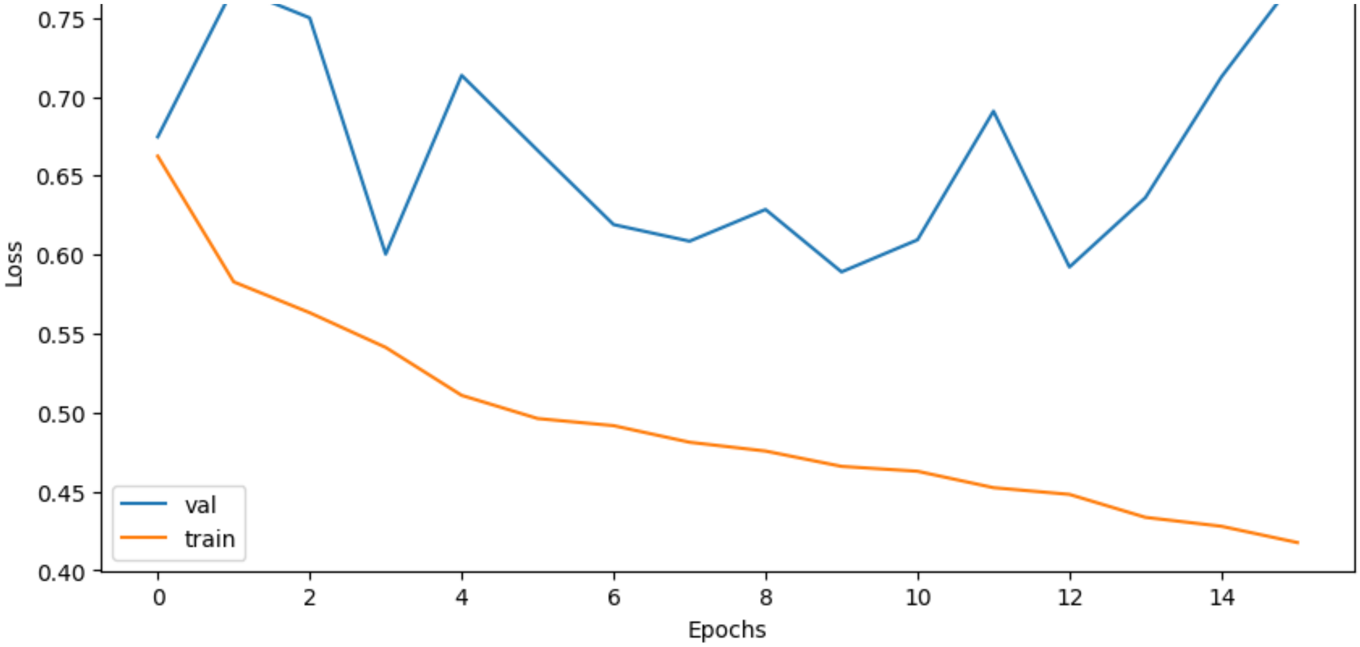


```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_adam_lr001.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
```

```
Epoch: 0, Train loss: 0.6623575282366698, Train accuracy: 0.6025869250297546
Val loss: 0.6745547907693045, Val accuracy: 0.5893665158371041
Epoch: 1, Train loss: 0.5827435219779358, Train accuracy: 0.6795473098754883
Val loss: 0.7694737911224365, Val accuracy: 0.41289592760180993
Epoch: 2, Train loss: 0.5632066259195271, Train accuracy: 0.7055780291557312
Val loss: 0.7499530996595111, Val accuracy: 0.4445701357466063
Epoch: 3, Train loss: 0.5412305806853064, Train accuracy: 0.7177041172981262
Val loss: 0.6003076093537466, Val accuracy: 0.7002262443438914
Epoch: 4, Train loss: 0.5108654524688289, Train accuracy: 0.7329021692276001
Val loss: 0.7135301913533892, Val accuracy: 0.6085972850678733
Epoch: 5, Train loss: 0.49618083702139826, Train accuracy: 0.748100221157074
Val loss: 0.6660270180021014, Val accuracy: 0.6165158371040724
Epoch: 6, Train loss: 0.49166618207817414, Train accuracy: 0.7443815469741821
Val loss: 0.6190203172819955, Val accuracy: 0.6481900452488688
Epoch: 7, Train loss: 0.48115725715312635, Train accuracy: 0.7532740235328674
Val loss: 0.6085034949438912, Val accuracy: 0.669683257918552
Epoch: 8, Train loss: 0.47557661750187485, Train accuracy: 0.7632982730865479
Val loss: 0.6285936151232038, Val accuracy: 0.6798642533936652
Epoch: 9, Train loss: 0.4658699411534454, Train accuracy: 0.7634599804878235
Val loss: 0.5890463846070426, Val accuracy: 0.6900452488687783
Epoch: 10, Train loss: 0.46278184839285663, Train accuracy: 0.767825365066528
Val loss: 0.6093596134866988, Val accuracy: 0.7002262443438914
Epoch: 11, Train loss: 0.4523827652345286, Train accuracy: 0.7744542956352234
Val loss: 0.6908440504755293, Val accuracy: 0.6346153846153846
Epoch: 12, Train loss: 0.4481271221258029, Train accuracy: 0.7729991674423218
Val loss: 0.5921272465160914, Val accuracy: 0.6877828054298643
Epoch: 13, Train loss: 0.43358854890929754, Train accuracy: 0.778658032417297
Val loss: 0.6360947319439479, Val accuracy: 0.668552036199095
Epoch: 14, Train loss: 0.4279327519978837, Train accuracy: 0.7940177917480469
Val loss: 0.7125345383371626, Val accuracy: 0.6561085972850679
Epoch: 15, Train loss: 0.41763163322677704, Train accuracy: 0.789490699768066
Val loss: 0.7760904942240033, Val accuracy: 0.6470588235294118
Early stopped training at epoch 15
```



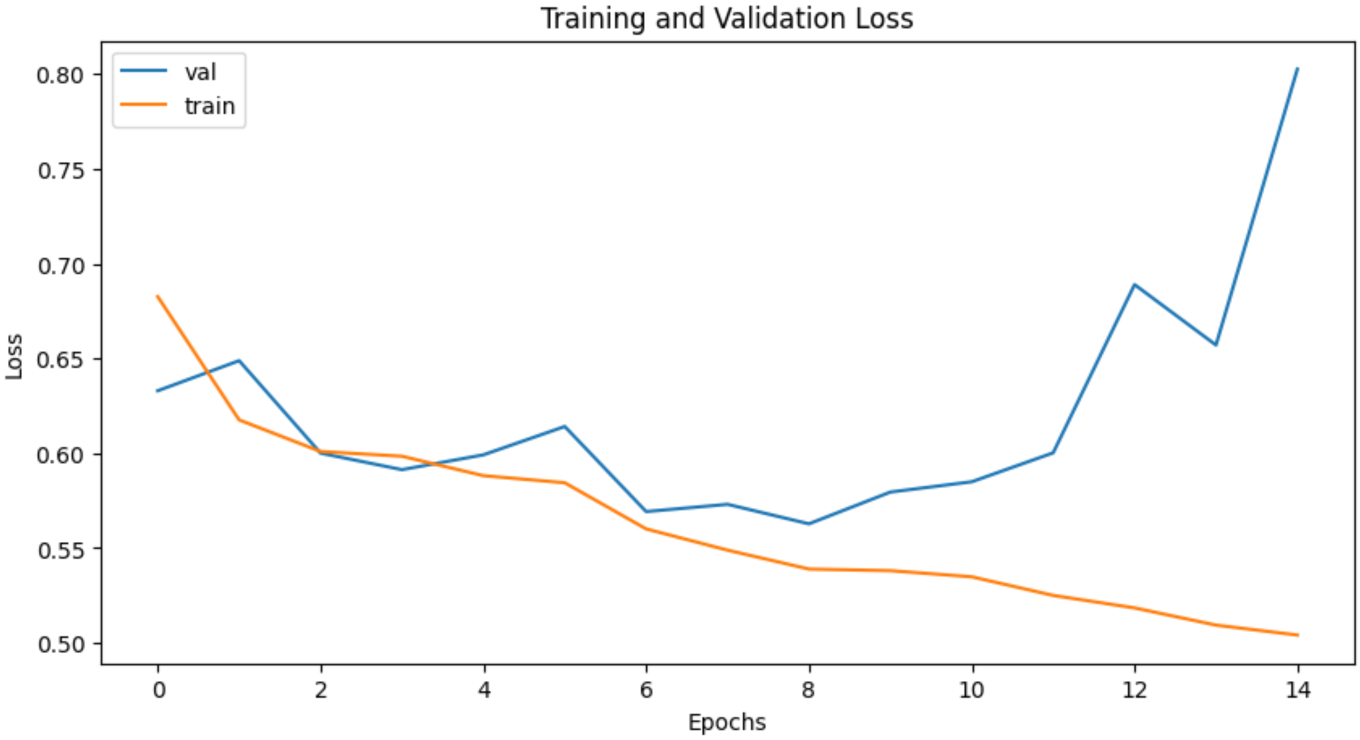
Training and Validation Loss



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 10 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_adam_lr001.pth
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
```

```
Epoch: 0, Train loss: 0.6825836147805645, Train accuracy: 0.5933993458747864
Val loss: 0.6329395771026611, Val accuracy: 0.5750577367205543
Epoch: 1, Train loss: 0.6175654076113559, Train accuracy: 0.631848156452179
Val loss: 0.648821873324258, Val accuracy: 0.5854503464203233
Epoch: 2, Train loss: 0.6009088196376763, Train accuracy: 0.6496699452400208
Val loss: 0.6000505600656783, Val accuracy: 0.6628175519630485
Epoch: 3, Train loss: 0.5984042203859135, Train accuracy: 0.6620461940765381
Val loss: 0.5913248317582267, Val accuracy: 0.6651270207852193
Epoch: 4, Train loss: 0.5881556952747181, Train accuracy: 0.6640263795852661
Val loss: 0.5991356117384775, Val accuracy: 0.6501154734411085
Epoch: 5, Train loss: 0.5844145541143889, Train accuracy: 0.672937273979187
Val loss: 0.6141006946563721, Val accuracy: 0.6685912240184757
```

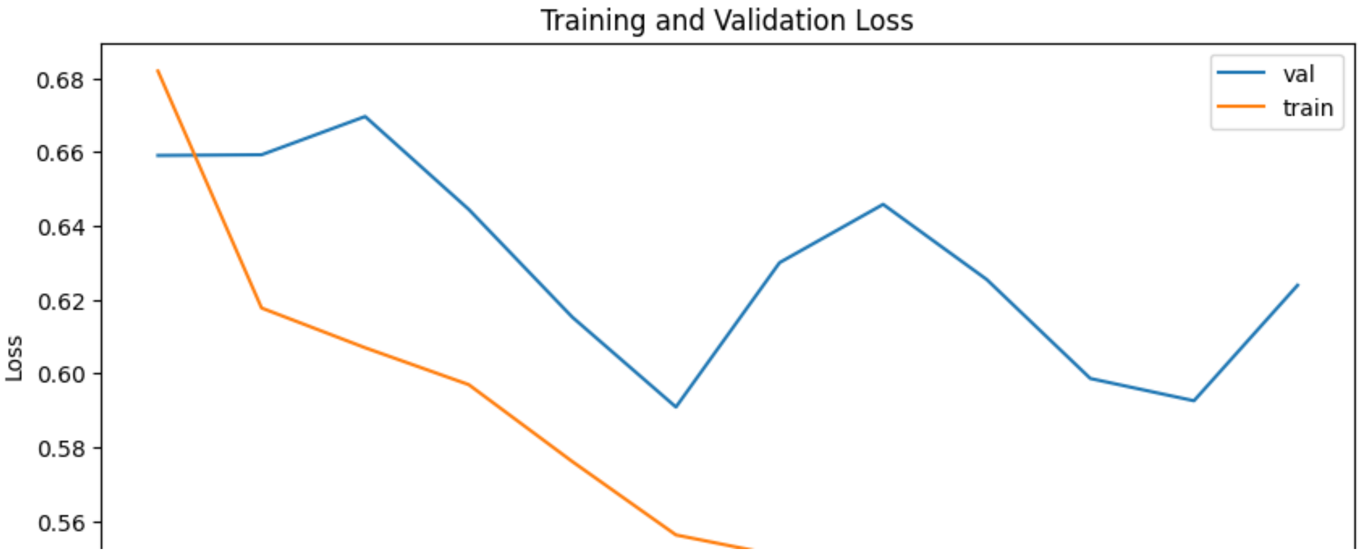
```
Epoch: 6, Train loss: 0.5600698428185467, Train accuracy: 0.6843234300613403
Val loss: 0.5692156979015895, Val accuracy: 0.6593533487297921
Epoch: 7, Train loss: 0.5488641889575291, Train accuracy: 0.6955445408821106
Val loss: 0.5730458583150592, Val accuracy: 0.6628175519630485
Epoch: 8, Train loss: 0.5389014973892237, Train accuracy: 0.6988449096679688
Val loss: 0.562750015939985, Val accuracy: 0.6674364896073903
Epoch: 9, Train loss: 0.5380438177892477, Train accuracy: 0.6973597407341003
Val loss: 0.579510544027601, Val accuracy: 0.6454965357967667
Epoch: 10, Train loss: 0.5348053332602624, Train accuracy: 0.7044554352760315
Val loss: 0.5849387560571944, Val accuracy: 0.6593533487297921
Epoch: 11, Train loss: 0.524986326163358, Train accuracy: 0.7082508206367493
Val loss: 0.6001993843487331, Val accuracy: 0.6131639722863741
Epoch: 12, Train loss: 0.5183976530635318, Train accuracy: 0.7140263915061951
Val loss: 0.6889567460332598, Val accuracy: 0.5577367205542725
Epoch: 13, Train loss: 0.509321390559571, Train accuracy: 0.7133663296699524
Val loss: 0.6570460711206708, Val accuracy: 0.5866050808314087
Epoch: 14, Train loss: 0.5040862974160575, Train accuracy: 0.7323432564735413
Val loss: 0.8026251196861266, Val accuracy: 0.5288683602771362
Early stopped training at epoch 14
```

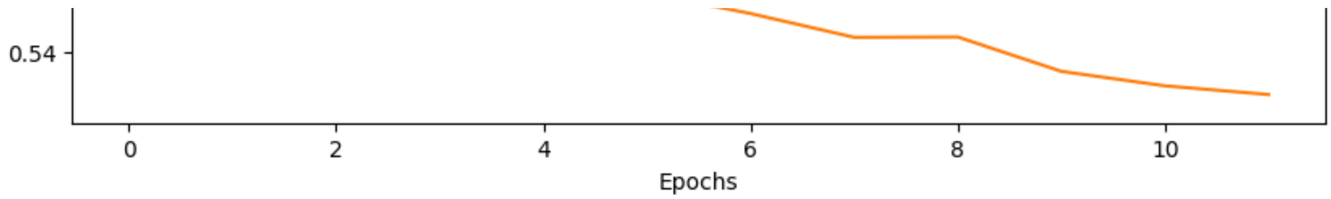


```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
```

```
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_adam_lr001.pth
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
```

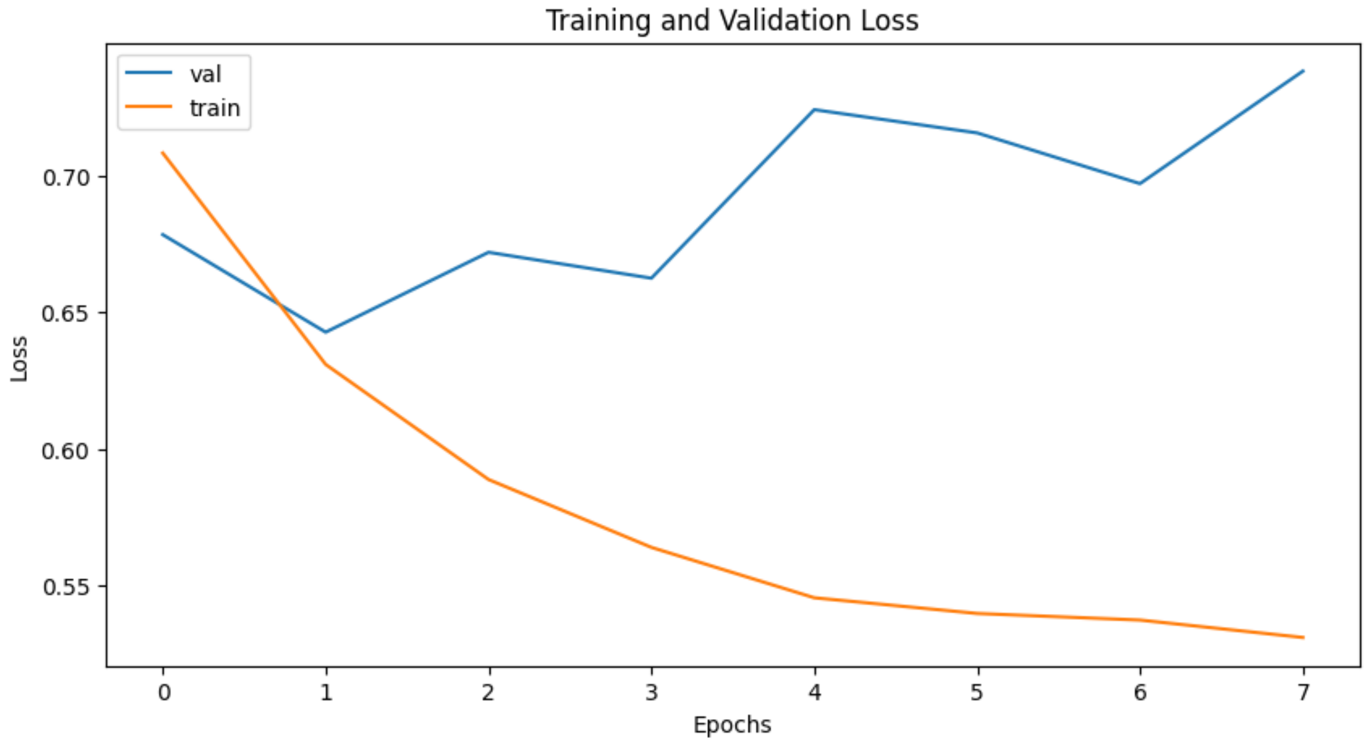
```
Epoch: 0, Train loss: 0.6819666133558165, Train accuracy: 0.5926975607872009
Val loss: 0.6590854270117624, Val accuracy: 0.632438739789965
Epoch: 1, Train loss: 0.617742661517157, Train accuracy: 0.6467155814170837
Val loss: 0.6592709251812526, Val accuracy: 0.632438739789965
Epoch: 2, Train loss: 0.6069676179454978, Train accuracy: 0.6575525403022766
Val loss: 0.6696439640862601, Val accuracy: 0.5670945157526255
Epoch: 3, Train loss: 0.5969698571014023, Train accuracy: 0.6650550365447998
Val loss: 0.6444673623357501, Val accuracy: 0.6301050175029171
Epoch: 4, Train loss: 0.5761125607067603, Train accuracy: 0.6750583648681641
Val loss: 0.6152532611574446, Val accuracy: 0.632438739789965
Epoch: 5, Train loss: 0.5562282881565037, Train accuracy: 0.6867288947105408
Val loss: 0.5908962999071393, Val accuracy: 0.6592765460910152
Epoch: 6, Train loss: 0.5506094128897128, Train accuracy: 0.6872290968894958
Val loss: 0.6300373503140041, Val accuracy: 0.6266044340723453
Epoch: 7, Train loss: 0.5441705519734084, Train accuracy: 0.6902300715446472
Val loss: 0.6458339606012617, Val accuracy: 0.6102683780630105
Epoch: 8, Train loss: 0.5442623536799025, Train accuracy: 0.6970657110214233
Val loss: 0.6254570824759347, Val accuracy: 0.6196032672112018
Epoch: 9, Train loss: 0.5349502865772877, Train accuracy: 0.6982327699661255
Val loss: 0.5986301728657314, Val accuracy: 0.6382730455075846
Epoch: 10, Train loss: 0.5310289678910686, Train accuracy: 0.7047349214553833
Val loss: 0.5925987958908081, Val accuracy: 0.6569428238039673
Epoch: 11, Train loss: 0.528671299707655, Train accuracy: 0.7027342319488525
Val loss: 0.6239062973431179, Val accuracy: 0.6114352392065344
Early stopped training at epoch 11
```





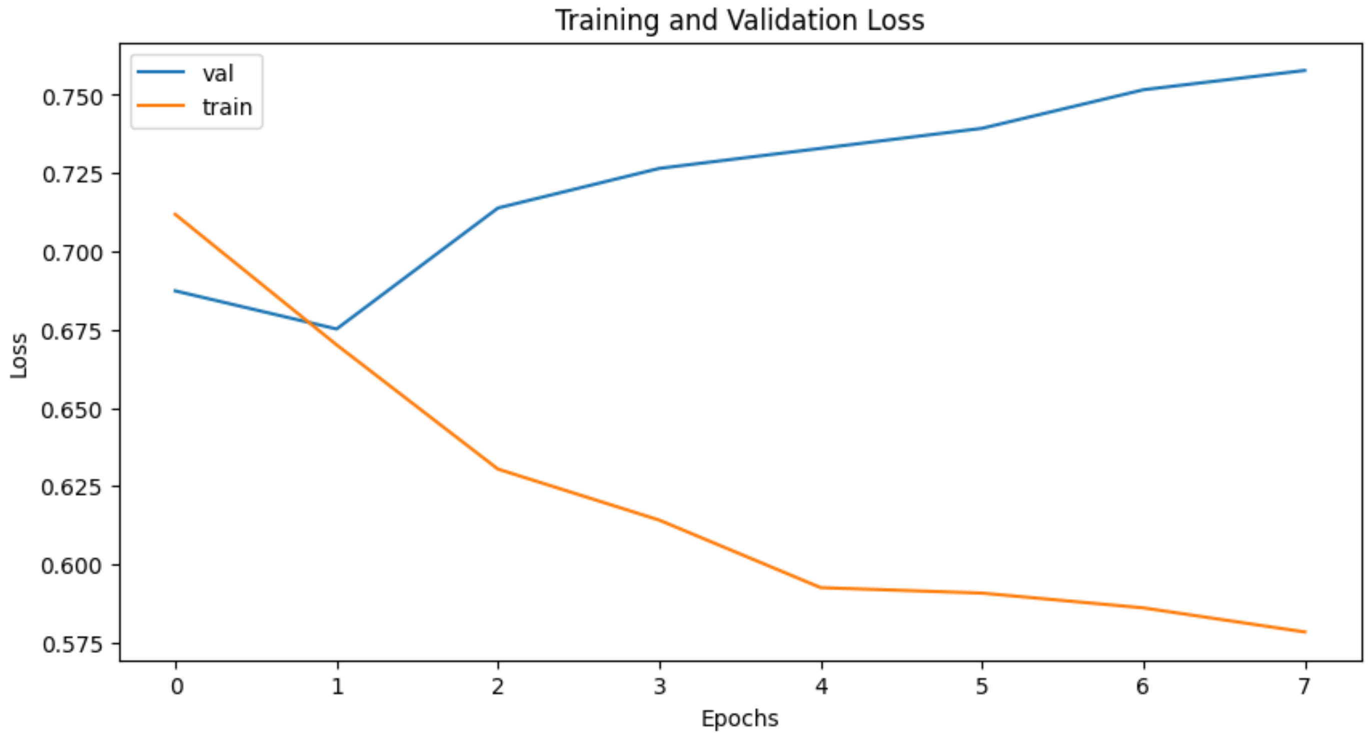
## batch size of 64, learning rate of 0.0001

```
train_loader_3min = load_data(HypoDataset(ecg_3min_train, art_3min_train, eeg_3mi
val_loader_3min = load_data(HypoDataset(ecg_3min_val, art_3min_val, eeg_3min_val,
test_loader_3min = load_data(HypoDataset(ecg_3min_test, art_3min_test, eeg_3min_t

train_loader_5min = load_data(HypoDataset(ecg_5min_train, art_5min_train, eeg_5mi
val_loader_5min = load_data(HypoDataset(ecg_5min_val, art_5min_val, eeg_5min_val,
test_loader_5min = load_data(HypoDataset(ecg_5min_test, art_5min_test, eeg_5min_t

train_loader_10min = load_data(HypoDataset(ecg_10min_train, art_10min_train, eeg_
val_loader_10min = load_data(HypoDataset(ecg_10min_val, art_10min_val, eeg_10min_
test_loader_10min = load_data(HypoDataset(ecg_10min_test, art_10min_test, eeg_10m

train_loader_15min = load_data(HypoDataset(ecg_15min_train, art_15min_train, eeg_
val_loader_15min = load_data(HypoDataset(ecg_15min_val, art_15min_val, eeg_15min_
test_loader_15min = load_data(HypoDataset(ecg_15min_test, art_15min_test, eeg_15m


loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_batch_64.pth"
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.708515659452862, Train accuracy: 0.5731707215309143
Val loss: 0.6785675329821451, Val accuracy: 0.5887640449438202
Epoch: 1, Train loss: 0.6309265550967206, Train accuracy: 0.6498716473579407
Val loss: 0.6427682893616812, Val accuracy: 0.6921348314606741
Epoch: 2, Train loss: 0.5886798926434254, Train accuracy: 0.6821244955062866
Val loss: 0.6720895128590719, Val accuracy: 0.6438202247191012
Epoch: 3, Train loss: 0.5637687385770873, Train accuracy: 0.7082798480987549
Val loss: 0.6625788169247764, Val accuracy: 0.6539325842696629
Epoch: 4, Train loss: 0.5452647931462534, Train accuracy: 0.7138960361480713
Val loss: 0.72444600718362, Val accuracy: 0.43820224719101125
Epoch: 5, Train loss: 0.5395059259620343, Train accuracy: 0.7257702350616455
Val loss: 0.7159190263066973, Val accuracy: 0.5134831460674157
Epoch: 6, Train loss: 0.5370843957569235, Train accuracy: 0.7209563255310059
Val loss: 0.6972964491162981, Val accuracy: 0.5674157303370787
Epoch: 7, Train loss: 0.5306925174513586, Train accuracy: 0.7265725135803223
Val loss: 0.7385896529470171, Val accuracy: 0.4438202247191011
Early stopped training at epoch 7
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5
```

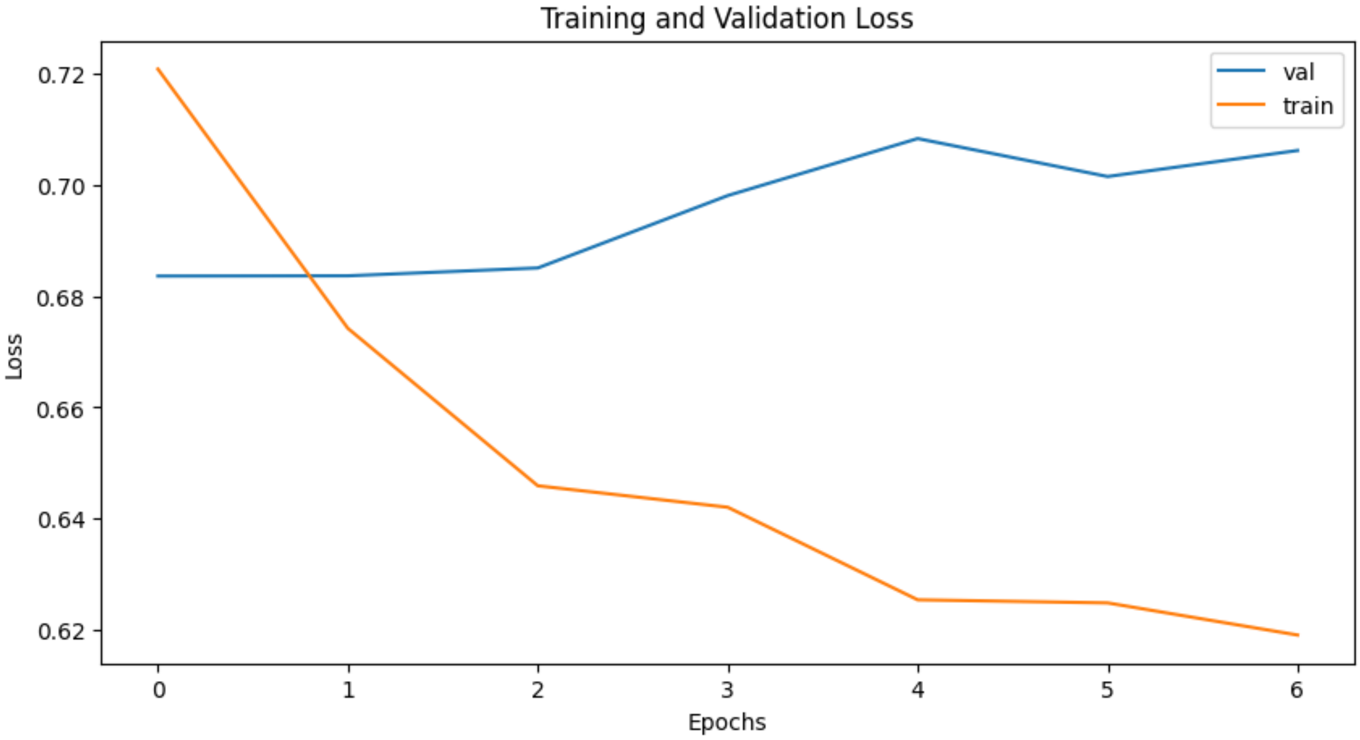
```
# 5 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_batch_64.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.7118410323643357, Train accuracy: 0.5513338446617126
Val loss: 0.6873951426574163, Val accuracy: 0.5769230769230769
Epoch: 1, Train loss: 0.6701529284687705, Train accuracy: 0.5869038105010986
Val loss: 0.6752147376537324, Val accuracy: 0.6278280542986425
Epoch: 2, Train loss: 0.6304483319735855, Train accuracy: 0.6323363184928894
Val loss: 0.7138725178582327, Val accuracy: 0.41855203619909503
Epoch: 3, Train loss: 0.6141451123653155, Train accuracy: 0.6572352647781372
Val loss: 0.7265264179025378, Val accuracy: 0.4117647058823529
Epoch: 4, Train loss: 0.5925726129147153, Train accuracy: 0.6680679321289062
Val loss: 0.7329202592372895, Val accuracy: 0.41289592760180993
Epoch: 5, Train loss: 0.5908209820765108, Train accuracy: 0.6742118000984192
Val loss: 0.7393313518592289, Val accuracy: 0.415158371040724
Epoch: 6, Train loss: 0.5861009271085118, Train accuracy: 0.671301543712616
Val loss: 0.751654211963926, Val accuracy: 0.41402714932126694
Epoch: 7, Train loss: 0.5784111587964631, Train accuracy: 0.6806790828704834
Val loss: 0.757819937808173, Val accuracy: 0.41289592760180993
Early stopped training at epoch 7
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5
```

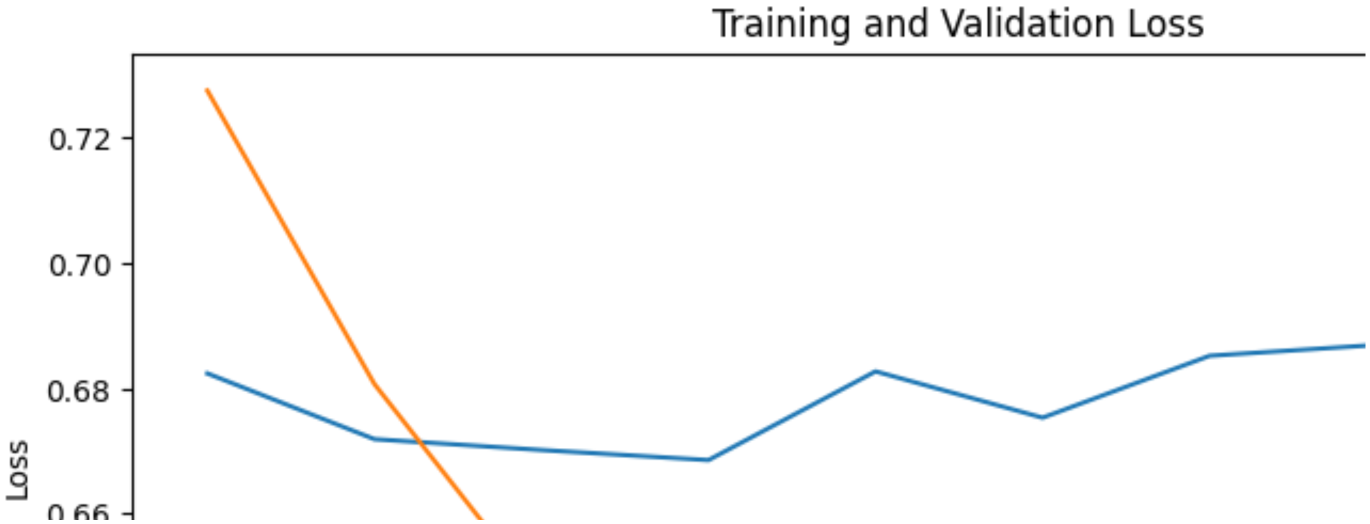
```
# 10 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_batch_64.pth"
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
```

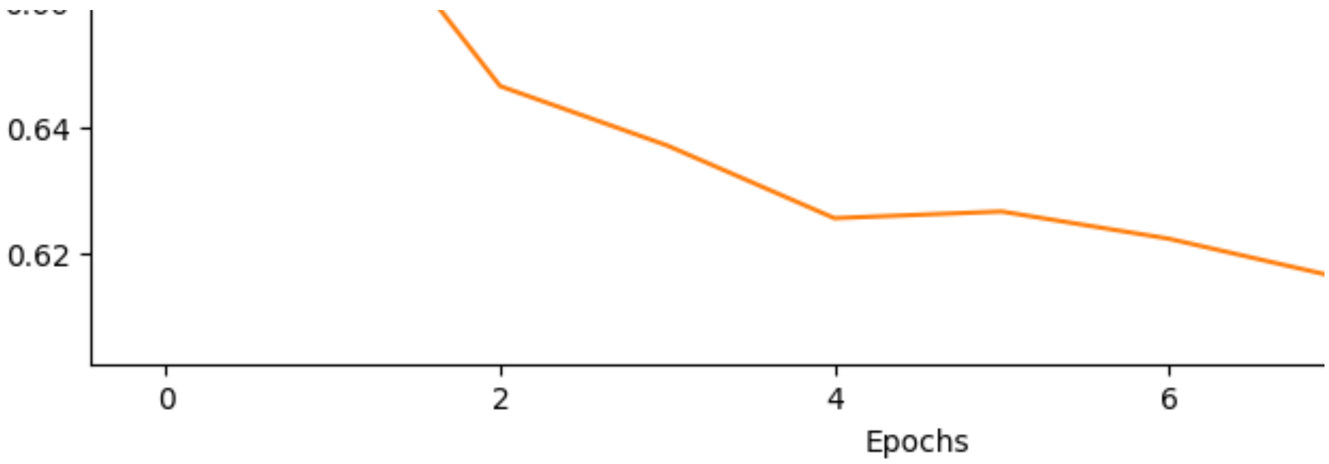
```
Epoch: 0, Train loss: 0.7208804701027697, Train accuracy: 0.5481848120689392
Val loss: 0.6836387131895338, Val accuracy: 0.5704387990762124
Epoch: 1, Train loss: 0.6742255520899304, Train accuracy: 0.593894362449646
Val loss: 0.6836815646716525, Val accuracy: 0.581986143187067
Epoch: 2, Train loss: 0.6458957396324712, Train accuracy: 0.6145214438438416
Val loss: 0.6850925854274204, Val accuracy: 0.5773672055427251
Epoch: 3, Train loss: 0.6420334300192276, Train accuracy: 0.6196369528770447
Val loss: 0.6981198361941746, Val accuracy: 0.4618937644341801
Epoch: 4, Train loss: 0.6253775739433742, Train accuracy: 0.6301980018615723
Val loss: 0.7083785576479775, Val accuracy: 0.4249422632794457З
Epoch: 5, Train loss: 0.6248230246427429, Train accuracy: 0.6303630471229553
Val loss: 0.7015470862388611, Val accuracy: 0.4515011547344111
Epoch: 6, Train loss: 0.6190382476293608, Train accuracy: 0.6407590508460999
Val loss: 0.7062313556671141, Val accuracy: 0.43648960739030024
Early stopped training at epoch 6
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_batch_64.pth"
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
```

```
Epoch: 0, Train loss: 0.727406423642024, Train accuracy: 0.565355122089386
Val loss: 0.6823473572731019, Val accuracy: 0.5670945157526255
Epoch: 1, Train loss: 0.6806540041814132, Train accuracy: 0.603367805480957
Val loss: 0.6718706403459821, Val accuracy: 0.5845974329054843
Epoch: 2, Train loss: 0.6466611295630432, Train accuracy: 0.6215404868125916
Val loss: 0.6701412839548928, Val accuracy: 0.6137689614935823
Epoch: 3, Train loss: 0.6371913159756153, Train accuracy: 0.6260420083999634
Val loss: 0.6685370547430856, Val accuracy: 0.6207701283547258
Epoch: 4, Train loss: 0.6256083709750823, Train accuracy: 0.6318773031234741
Val loss: 0.6826680387769427, Val accuracy: 0.5950991831971996
Epoch: 5, Train loss: 0.6266731446327548, Train accuracy: 0.6337112188339233
Val loss: 0.6752888815743583, Val accuracy: 0.6102683780630105
Epoch: 6, Train loss: 0.6223405590849187, Train accuracy: 0.6432144045829773
Val loss: 0.6851547658443451, Val accuracy: 0.574095682613769
Epoch: 7, Train loss: 0.6162802856260238, Train accuracy: 0.6448816061019897
Val loss: 0.6868826448917389, Val accuracy: 0.5787631271878646
Epoch: 8, Train loss: 0.6152239298574047, Train accuracy: 0.6457152366638184
Val loss: 0.6912671157291957, Val accuracy: 0.5192532088681447
Epoch: 9, Train loss: 0.6083496601512409, Train accuracy: 0.6487162113189697
Val loss: 0.6870117613247462, Val accuracy: 0.558926487747958
Early stopped training at epoch 9
```



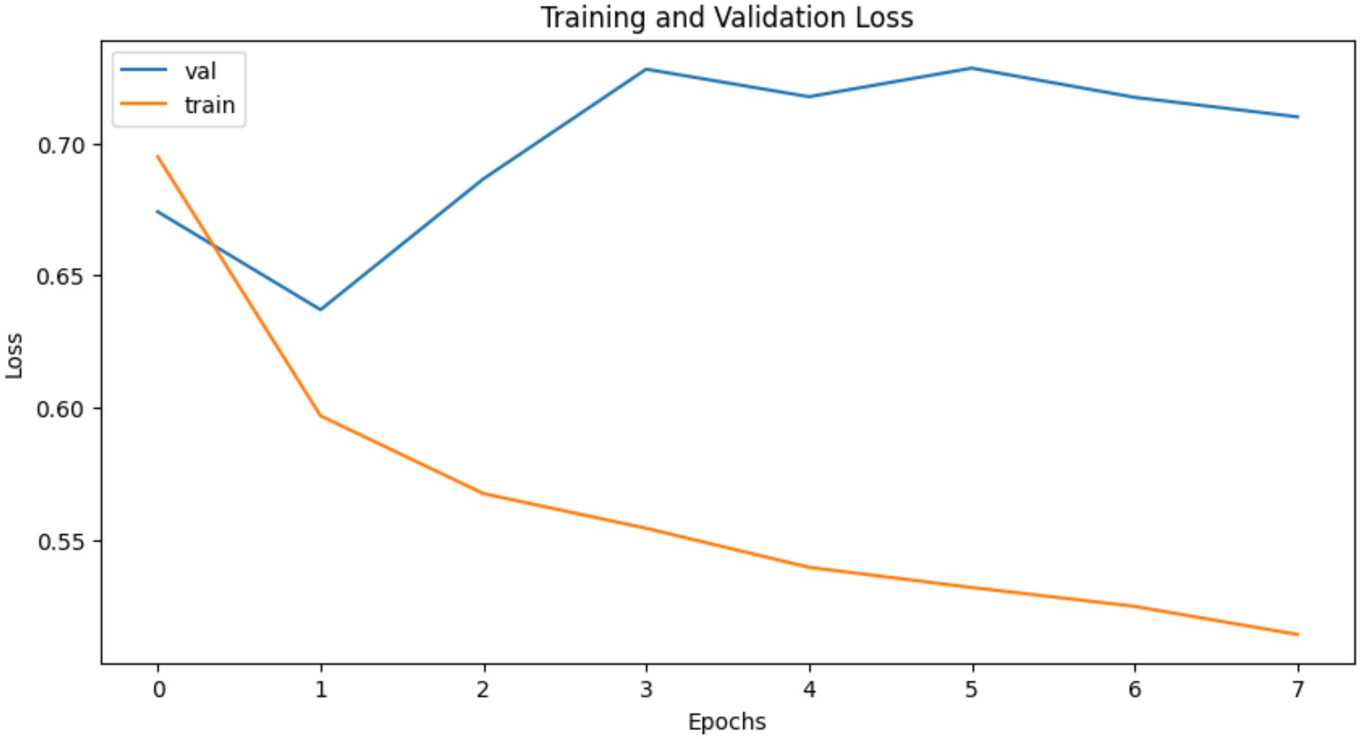


## batch size of 32, learning rate of 0.0001

```
train_loader_3min = load_data(HypoDataset(ecg_3min_train, art_3min_train, eeg_3mi
val_loader_3min = load_data(HypoDataset(ecg_3min_val, art_3min_val, eeg_3min_val,
test_loader_3min = load_data(HypoDataset(ecg_3min_test, art_3min_test, eeg_3min_t

train_loader_5min = load_data(HypoDataset(ecg_5min_train, art_5min_train, eeg_5mi
val_loader_5min = load_data(HypoDataset(ecg_5min_val, art_5min_val, eeg_5min_val,
test_loader_5min = load_data(HypoDataset(ecg_5min_test, art_5min_test, eeg_5min_t

train_loader_10min = load_data(HypoDataset(ecg_10min_train, art_10min_train, eeg_
val_loader_10min = load_data(HypoDataset(ecg_10min_val, art_10min_val, eeg_10min_
test_loader_10min = load_data(HypoDataset(ecg_10min_test, art_10min_test, eeg_10m

train_loader_15min = load_data(HypoDataset(ecg_15min_train, art_15min_train, eeg_
val_loader_15min = load_data(HypoDataset(ecg_15min_val, art_15min_val, eeg_15min_
test_loader_15min = load_data(HypoDataset(ecg_15min_test, art_15min_test, eeg_15m


loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_batch_32.pth"
```

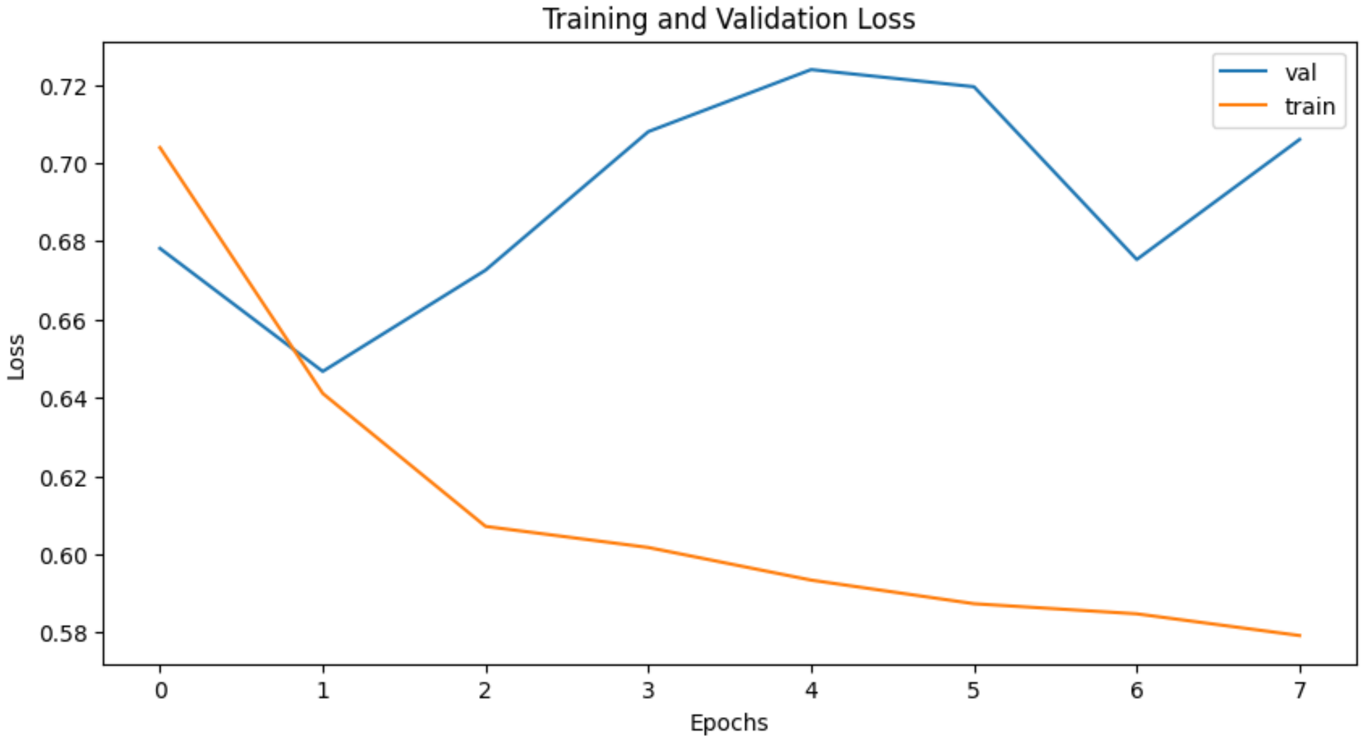
```
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
```

```
Epoch: 0, Train loss: 0.6948849225227884, Train accuracy: 0.5893774032592773
Val loss: 0.6740936913660593, Val accuracy: 0.6224719101123596
Epoch: 1, Train loss: 0.5969796575538919, Train accuracy: 0.6742618680000305
Val loss: 0.6370618109192167, Val accuracy: 0.6966292134831461
Epoch: 2, Train loss: 0.567604552643596, Train accuracy: 0.7058728933334351
Val loss: 0.6864937948329107, Val accuracy: 0.5719101123595506
Epoch: 3, Train loss: 0.5544919323400906, Train accuracy: 0.7118099927902222
Val loss: 0.7279485974993025, Val accuracy: 0.42696629213483145
Epoch: 4, Train loss: 0.5397531432272993, Train accuracy: 0.7246469855308533
Val loss: 0.717547527381352, Val accuracy: 0.47415730337078654
Epoch: 5, Train loss: 0.5321106791725819, Train accuracy: 0.7289794683456421
Val loss: 0.7283448491777694, Val accuracy: 0.47752808988764045
Epoch: 6, Train loss: 0.5249905674234129, Train accuracy: 0.7293003797531128
Val loss: 0.7173200121947697, Val accuracy: 0.5067415730337078
Epoch: 7, Train loss: 0.5143601030532141, Train accuracy: 0.733793318271637
Val loss: 0.709942741053445, Val accuracy: 0.5662921348314607
Early stopped training at epoch 7
```



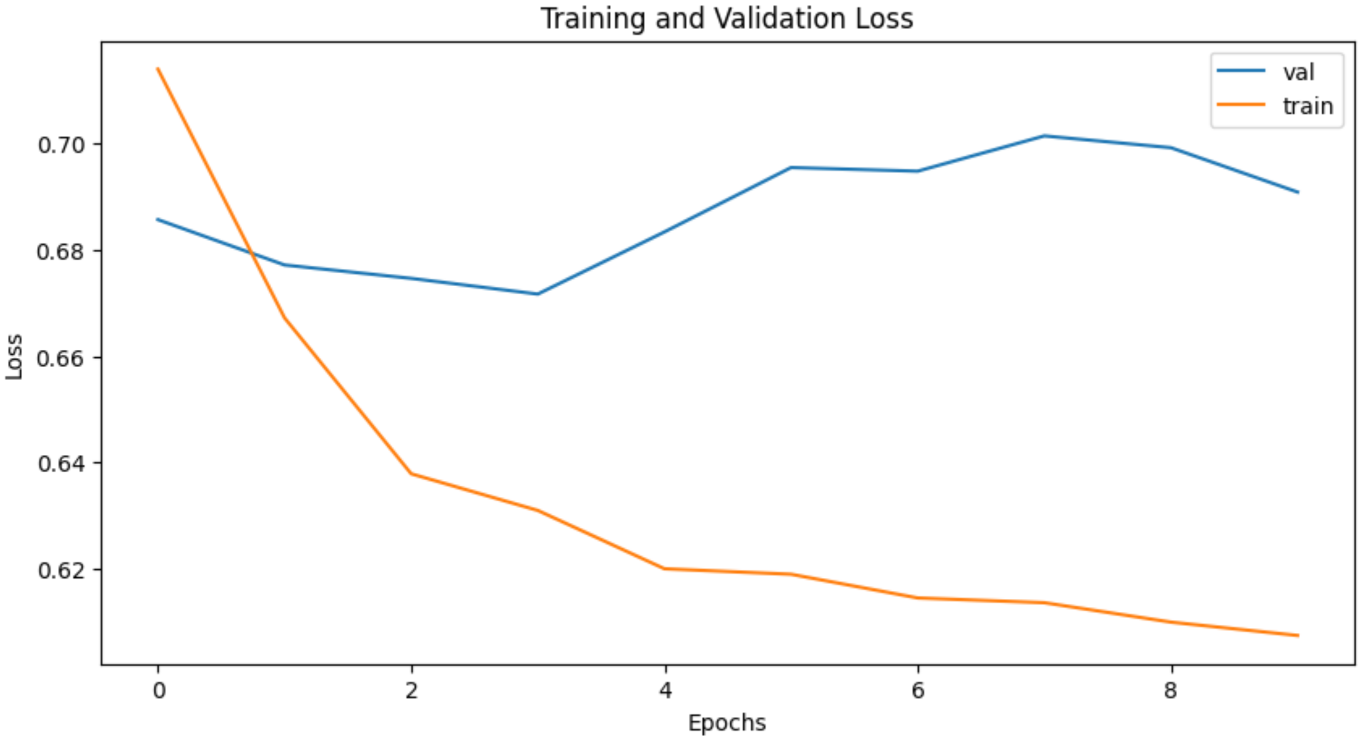
```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
```

```
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_batch_32.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.7039331664746099, Train accuracy: 0.5775262713432312
Val loss: 0.6781749108007974, Val accuracy: 0.582579185520362
Epoch: 1, Train loss: 0.6411088988389715, Train accuracy: 0.6328213214874268
Val loss: 0.6467356149639402, Val accuracy: 0.6821266968325792
Epoch: 2, Train loss: 0.6071288484378312, Train accuracy: 0.6567502021789551
Val loss: 0.6726129800081253, Val accuracy: 0.6029411764705882
Epoch: 3, Train loss: 0.6017379299131438, Train accuracy: 0.6643492579460144
Val loss: 0.7080088342939106, Val accuracy: 0.4479638009049774
Epoch: 4, Train loss: 0.593398455312971, Train accuracy: 0.6693613529205322
Val loss: 0.7238584842000689, Val accuracy: 0.416289592760181
Epoch: 5, Train loss: 0.5873692816847644, Train accuracy: 0.6695230603218079
Val loss: 0.7194752799613136, Val accuracy: 0.4287330316742081
Epoch: 6, Train loss: 0.5847852636559316, Train accuracy: 0.6753435730934143
Val loss: 0.6753329421792712, Val accuracy: 0.5938914027149321
Epoch: 7, Train loss: 0.579234873554643, Train accuracy: 0.6840744018554688
Val loss: 0.7059995659760067, Val accuracy: 0.5248868778280543
Early stopped training at epoch 7
```



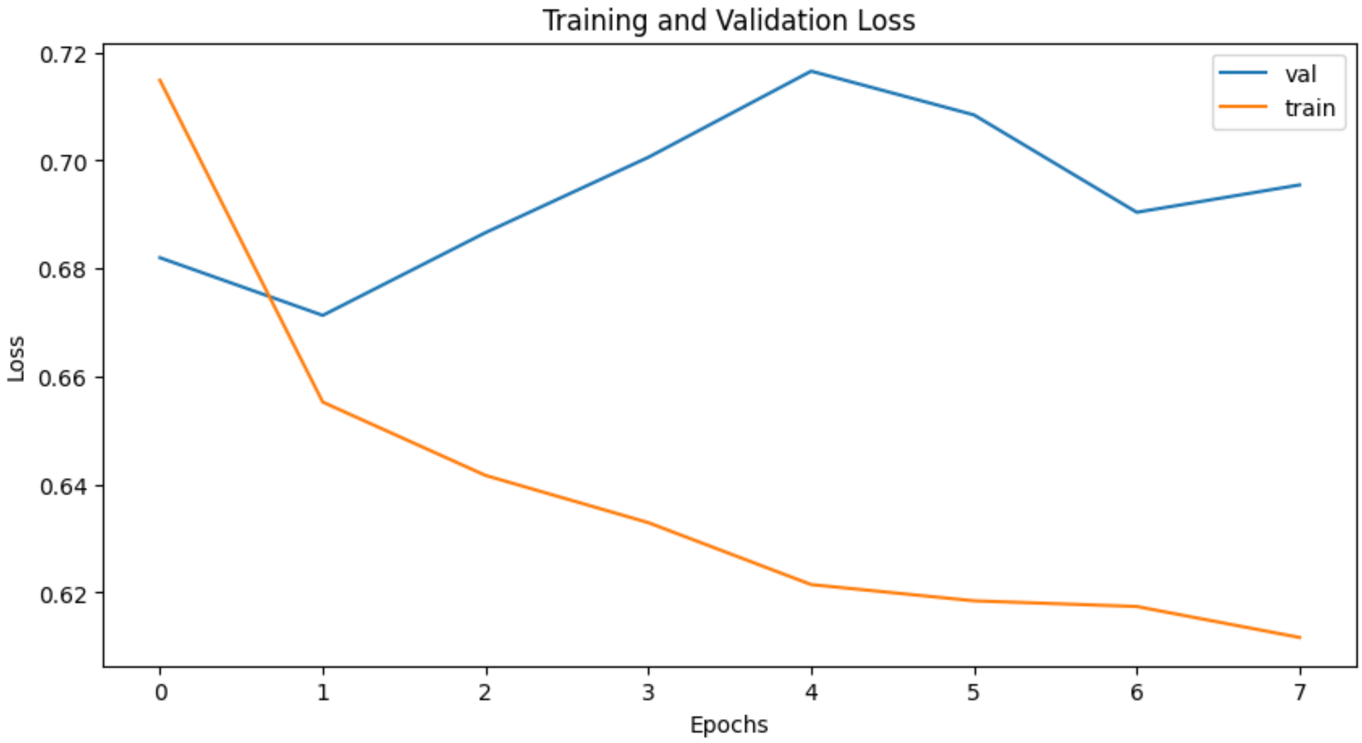
```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5
```

```
# 10 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_batch_32.pth"
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
Epoch: 0, Train loss: 0.7140539271996753, Train accuracy: 0.5612211227416992
Val loss: 0.6857216081448964, Val accuracy: 0.5681293302540416
Epoch: 1, Train loss: 0.6672189221130346, Train accuracy: 0.6141914129257202
Val loss: 0.6771831278290068, Val accuracy: 0.6247113163972287
Epoch: 2, Train loss: 0.6378718360029038, Train accuracy: 0.621947169303894
Val loss: 0.6746602399008614, Val accuracy: 0.6143187066974596
Epoch: 3, Train loss: 0.6309706899199156, Train accuracy: 0.6278877854347229
Val loss: 0.6717044562101364, Val accuracy: 0.6004618937644342
Epoch: 4, Train loss: 0.6199909397281043, Train accuracy: 0.6404290199279785
Val loss: 0.6834082241569246, Val accuracy: 0.5554272517321016
Epoch: 5, Train loss: 0.6189579978240992, Train accuracy: 0.644059419631958
Val loss: 0.6955280814852032, Val accuracy: 0.45842956120092376
Epoch: 6, Train loss: 0.6144947067739153, Train accuracy: 0.6336633563041687
Val loss: 0.6948439308575222, Val accuracy: 0.5023094688221709
Epoch: 7, Train loss: 0.6136021710268341, Train accuracy: 0.6429042816162109
Val loss: 0.7014869855982917, Val accuracy: 0.42725173210161665
Epoch: 8, Train loss: 0.6099446834510702, Train accuracy: 0.6455445289611816
Val loss: 0.6992634002651487, Val accuracy: 0.4491916859122402
Epoch: 9, Train loss: 0.6074326664307723, Train accuracy: 0.6551154851913452
Val loss: 0.6909108864409583, Val accuracy: 0.5011547344110855
Early stopped training at epoch 9
```



```
loss_func = nn.BCEWithLogitsLoss()
```

```
lr = 0.0001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_batch_32.pth"
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
Epoch: 0, Train loss: 0.7148341080394337, Train accuracy: 0.5590196847915649
Val loss: 0.6819890516775626, Val accuracy: 0.5822637106184364
Epoch: 1, Train loss: 0.6552626642039237, Train accuracy: 0.6043681502342224
Val loss: 0.6713173389434814, Val accuracy: 0.6406067677946324
Epoch: 2, Train loss: 0.6416835381055364, Train accuracy: 0.6303768157958984
Val loss: 0.6866374170338666, Val accuracy: 0.5507584597432905
Epoch: 3, Train loss: 0.6329421587052366, Train accuracy: 0.6315438747406006
Val loss: 0.700581619033107, Val accuracy: 0.47374562427071176
Epoch: 4, Train loss: 0.621481829462309, Train accuracy: 0.6373791098594666
Val loss: 0.7165269586775037, Val accuracy: 0.43990665110851807
Epoch: 5, Train loss: 0.6184727730215211, Train accuracy: 0.6400466561317444
Val loss: 0.7084236873520745, Val accuracy: 0.4340723453908985
Epoch: 6, Train loss: 0.6174213329964219, Train accuracy: 0.6395465135574341
Val loss: 0.6903981544353344, Val accuracy: 0.5227537922987164
Epoch: 7, Train loss: 0.6116877038226838, Train accuracy: 0.6473824381828308
Val loss: 0.695464708186962, Val accuracy: 0.5122520420070011
Early stopped training at epoch 7
```

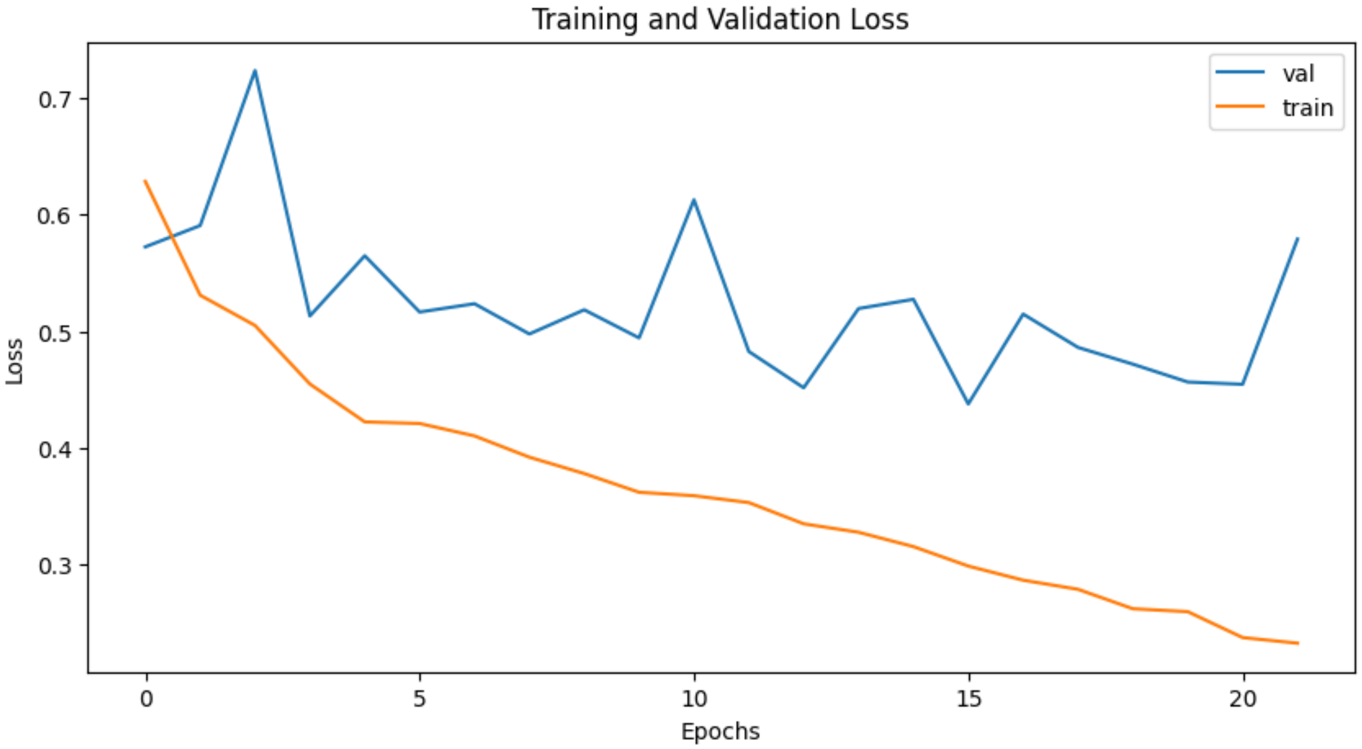


## batch size of 64, learning rate of 0.001

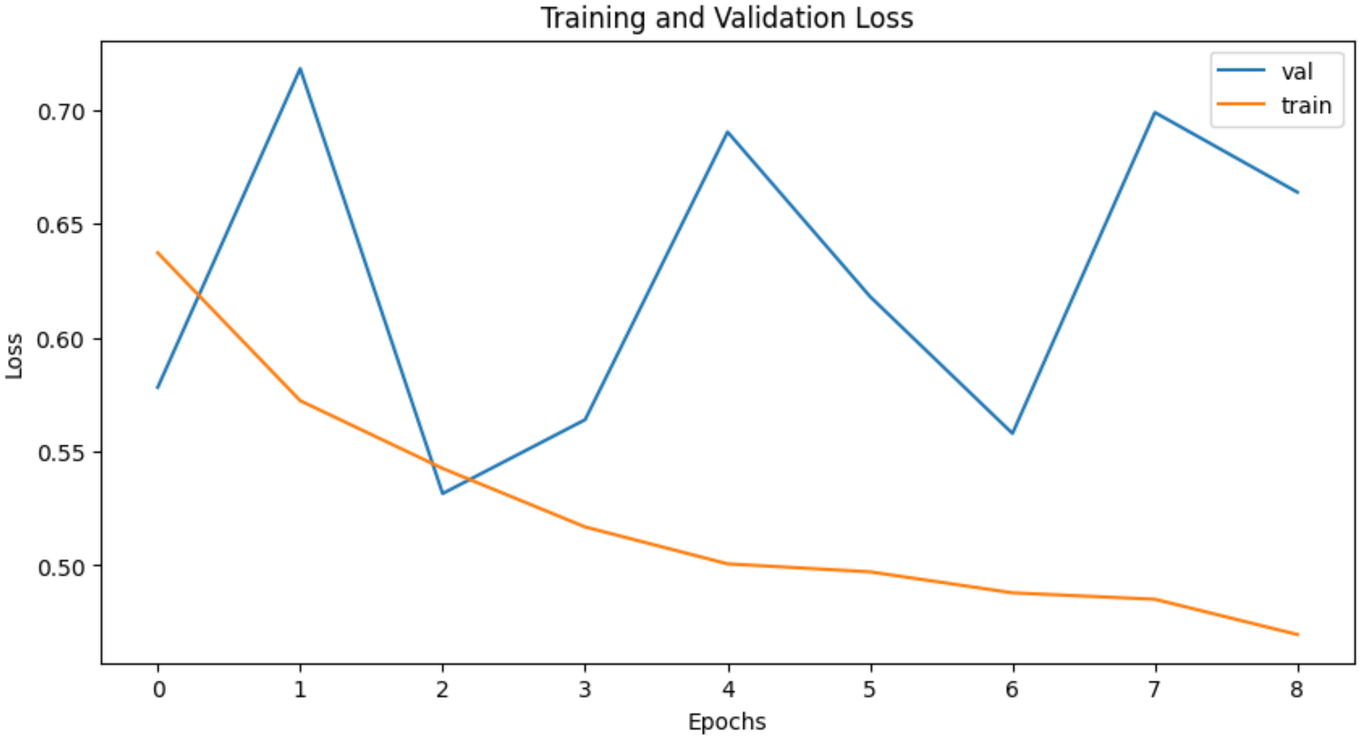
```
train_loader_3min = load_data(HypoDataset(ecg_3min_train, art_3min_train, eeg_3mi
val_loader_3min = load_data(HypoDataset(ecg_3min_val, art_3min_val, eeg_3min_val,
test_loader_3min = load_data(HypoDataset(ecg_3min_test, art_3min_test, eeg_3min_t

train_loader_5min = load_data(HypoDataset(ecg_5min_train, art_5min_train, eeg_5mi
val_loader_5min = load_data(HypoDataset(ecg_5min_val, art_5min_val, eeg_5min_val,
test_loader_5min = load_data(HypoDataset(ecg_5min_test, art_5min_test, eeg_5min_t

train_loader_10min = load_data(HypoDataset(ecg_10min_train, art_10min_train, eeg_
val_loader_10min = load_data(HypoDataset(ecg_10min_val, art_10min_val, eeg_10min_
test_loader_10min = load_data(HypoDataset(ecg_10min_test, art_10min_test, eeg_10m

train_loader_15min = load_data(HypoDataset(ecg_15min_train, art_15min_train, eeg_
val_loader_15min = load_data(HypoDataset(ecg_15min_val, art_15min_val, eeg_15min_
test_loader_15min = load_data(HypoDataset(ecg_15min_test, art_15min_test, eeg_15m


loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_batch_64_lr001.
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
```

```
    Epoch: 0, Train loss: 0.6286288557600455, Train accuracy: 0.6542041301727295
    Val loss: 0.5724362454244069, Val accuracy: 0.6719101123595506
    Epoch: 1, Train loss: 0.5309498315062541, Train accuracy: 0.7293003797531128
    Val loss: 0.590752688901765, Val accuracy: 0.6617977528089888
    Epoch: 2, Train loss: 0.5049286411264589, Train accuracy: 0.7511232495307922
    Val loss: 0.7238976210355759, Val accuracy: 0.6786516853932584
    Epoch: 3, Train loss: 0.4546609235896379, Train accuracy: 0.7772785425186157
    Val loss: 0.5130853142057146, Val accuracy: 0.7640449438202247
    Epoch: 4, Train loss: 0.4221179009019487, Train accuracy: 0.8029525279998779
    Val loss: 0.5648096565689359, Val accuracy: 0.7460674157303371
    Epoch: 5, Train loss: 0.42073197163574505, Train accuracy: 0.8048780560493469
    Val loss: 0.5164893588849476, Val accuracy: 0.7629213483146068
    Epoch: 6, Train loss: 0.410079524322713, Train accuracy: 0.8119384050369263
    Val loss: 0.5236147642135619, Val accuracy: 0.7617977528089888
    Epoch: 7, Train loss: 0.39180587547734397, Train accuracy: 0.8170731663703918
    Val loss: 0.49761974385806484, Val accuracy: 0.750561797752809
    Epoch: 8, Train loss: 0.37774009415365767, Train accuracy: 0.8275032043457031
    Val loss: 0.5183824939387184, Val accuracy: 0.750561797752809
```

```
Epoch: 9, Train loss: 0.3616632842512584, Train accuracy: 0.8340821862220764
Val loss: 0.4942594638892582, Val accuracy: 0.7730337078651686
Epoch: 10, Train loss: 0.3587183587167015, Train accuracy: 0.8385751247406006
Val loss: 0.6128867928470885, Val accuracy: 0.7640449438202247
Epoch: 11, Train loss: 0.3528036265914951, Train accuracy: 0.8430680632591248
Val loss: 0.4825763319219862, Val accuracy: 0.7910112359550562
Epoch: 12, Train loss: 0.3345688137683697, Train accuracy: 0.8515725135803223
Val loss: 0.4514357383762087, Val accuracy: 0.7955056179775281
Epoch: 13, Train loss: 0.3273255993374199, Train accuracy: 0.8613607287406921
Val loss: 0.5194730865103857, Val accuracy: 0.7797752808988764
Epoch: 14, Train loss: 0.31500477972171426, Train accuracy: 0.868741989135742
Val loss: 0.5273979838405337, Val accuracy: 0.7887640449438202
Epoch: 15, Train loss: 0.2982902011149043, Train accuracy: 0.8701861500740051
Val loss: 0.43752363111291614, Val accuracy: 0.8056179775280898
Epoch: 16, Train loss: 0.28616473549290705, Train accuracy: 0.879332482814788
Val loss: 0.5147169530391693, Val accuracy: 0.7842696629213484
Epoch: 17, Train loss: 0.27832890367936414, Train accuracy: 0.883344054222106
Val loss: 0.48608622380665367, Val accuracy: 0.7910112359550562
Epoch: 18, Train loss: 0.2616214782037845, Train accuracy: 0.8916880488395691
Val loss: 0.47168885597160887, Val accuracy: 0.8
Epoch: 19, Train loss: 0.2591321758725065, Train accuracy: 0.895057737827301
Val loss: 0.45637210564953934, Val accuracy: 0.7932584269662921
Epoch: 20, Train loss: 0.23688548782288033, Train accuracy: 0.905006408691406
Val loss: 0.45443618723324364, Val accuracy: 0.8044943820224719
Epoch: 21, Train loss: 0.232048218455333, Train accuracy: 0.9086970686912537
Val loss: 0.5791320651769638, Val accuracy: 0.7595505617977528
Early stopped training at epoch 21
```

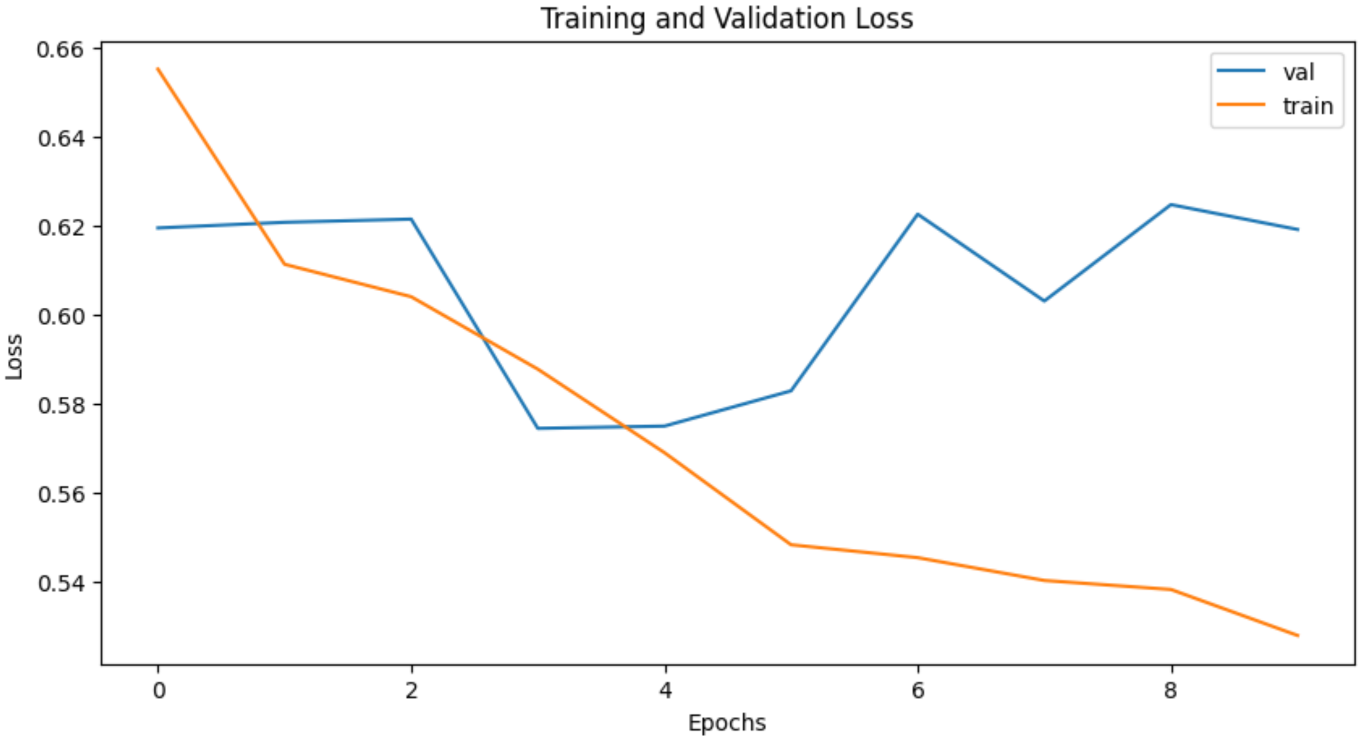


```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_batch_64_lr001.
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.6372702360538677, Train accuracy: 0.6231204271316528
Val loss: 0.578183942607471, Val accuracy: 0.6957013574660633
Epoch: 1, Train loss: 0.5723153434266173, Train accuracy: 0.6831042766571045
Val loss: 0.7182085301194872, Val accuracy: 0.5780542986425339
Epoch: 2, Train loss: 0.5425550641315834, Train accuracy: 0.710590124130249
Val loss: 0.5315225741692952, Val accuracy: 0.7126696832579186
Epoch: 3, Train loss: 0.516862752973407, Train accuracy: 0.7222312092781067
Val loss: 0.5639624808515822, Val accuracy: 0.6979638009049773
Epoch: 4, Train loss: 0.5006470910126679, Train accuracy: 0.7377526164054871
Val loss: 0.6903746809278216, Val accuracy: 0.6504524886877828
Epoch: 5, Train loss: 0.49713901750677136, Train accuracy: 0.7369441986083984
Val loss: 0.618091604539326, Val accuracy: 0.6742081447963801
Epoch: 6, Train loss: 0.4879147828733526, Train accuracy: 0.7521422505378723
Val loss: 0.5579406108175005, Val accuracy: 0.7149321266968326
Epoch: 7, Train loss: 0.4850919339525767, Train accuracy: 0.7529506683349609
Val loss: 0.6989466888563973, Val accuracy: 0.6300904977375565
Epoch: 8, Train loss: 0.46959454222080105, Train accuracy: 0.7584478855133057
Val loss: 0.6639024870736259, Val accuracy: 0.6380090497737556
Early stopped training at epoch 8
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
```

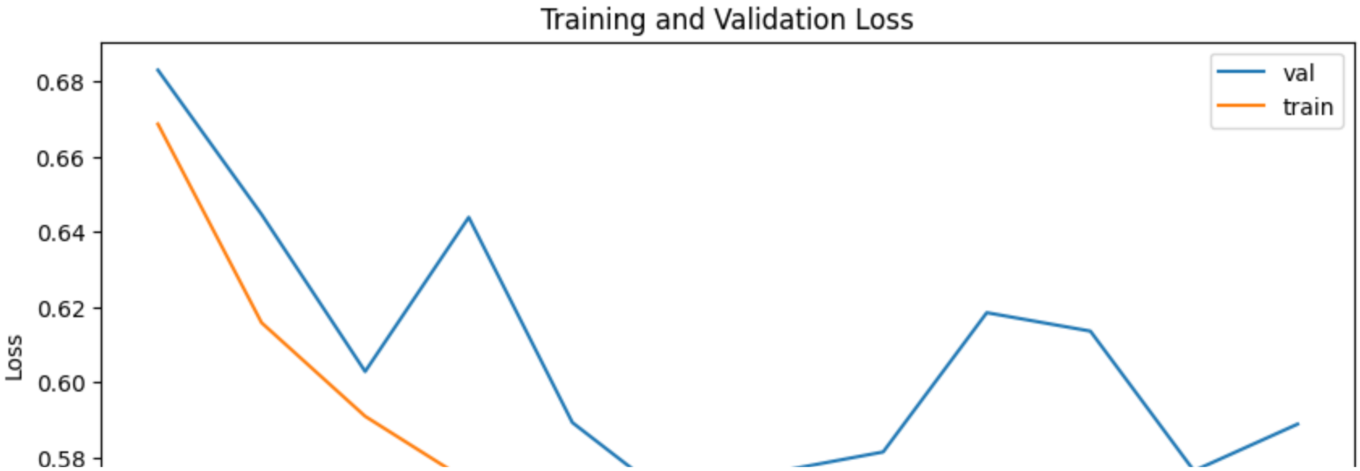
```
num_epoch = 100
early_stop_thresh = 5

# 10 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_batch_64_lr001
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
Epoch: 0, Train loss: 0.6551498413085938, Train accuracy: 0.6122112274169922
Val loss: 0.6194315808159965, Val accuracy: 0.6454965357967667
Epoch: 1, Train loss: 0.6112623969320417, Train accuracy: 0.6499999761581421
Val loss: 0.6207016238144465, Val accuracy: 0.6524249422632794
Epoch: 2, Train loss: 0.6039635260506432, Train accuracy: 0.6533003449440002
Val loss: 0.6214119834559304, Val accuracy: 0.6628175519630485
Epoch: 3, Train loss: 0.5876718424727815, Train accuracy: 0.6666666865348816
Val loss: 0.5743525922298433, Val accuracy: 0.6616628175519631
Epoch: 4, Train loss: 0.5688908403462702, Train accuracy: 0.6818481683731079
Val loss: 0.5748621587242398, Val accuracy: 0.6501154734411085
Epoch: 5, Train loss: 0.5481746702697804, Train accuracy: 0.6891089081764221
Val loss: 0.5827978423663549, Val accuracy: 0.6547344110854504
Epoch: 6, Train loss: 0.5452763848178851, Train accuracy: 0.6996699571609497
Val loss: 0.6225294087614333, Val accuracy: 0.6200923787528868
Epoch: 7, Train loss: 0.5401436352297024, Train accuracy: 0.6991749405860901
Val loss: 0.6029953147683825, Val accuracy: 0.6270207852193995
Epoch: 8, Train loss: 0.5381181974615594, Train accuracy: 0.6919142007827759
Val loss: 0.6246800422668456, Val accuracy: 0.6212471131639723
Epoch: 9, Train loss: 0.5277748222398286, Train accuracy: 0.7059406042098999
Val loss: 0.6190875428063528, Val accuracy: 0.6327944572748267
Early stopped training at epoch 9
```

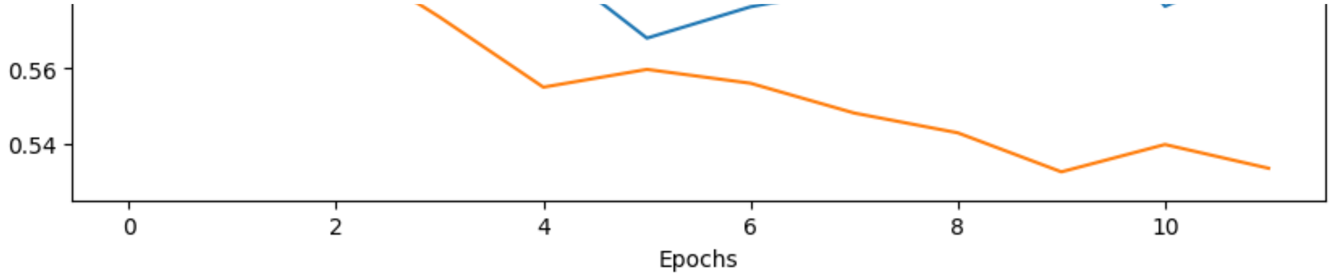


```
loss_func = nn.BCEWithLogitsLoss()
```

```
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_batch_64_lr001
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
```

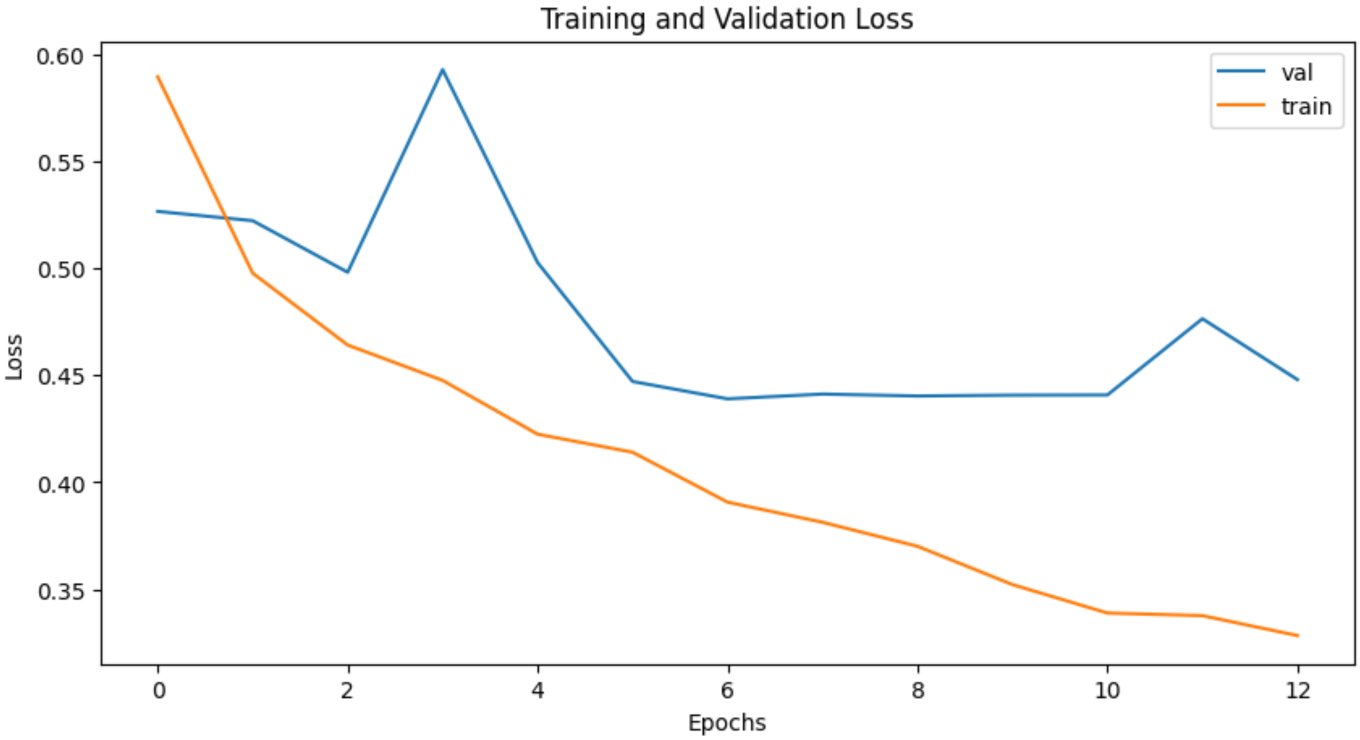
```
    Epoch: 0, Train loss: 0.6686201651082829, Train accuracy: 0.6073691248893738
    Val loss: 0.6829502667699542, Val accuracy: 0.5519253208868145
    Epoch: 1, Train loss: 0.6158484715388274, Train accuracy: 0.6468822956085205
    Val loss: 0.6446236414568766, Val accuracy: 0.6499416569428238
    Epoch: 2, Train loss: 0.5909857155880001, Train accuracy: 0.6710570454597473
    Val loss: 0.6028791410582406, Val accuracy: 0.6697782963827305
    Epoch: 3, Train loss: 0.5736186511717705, Train accuracy: 0.6758919358253479
    Val loss: 0.6438367281641278, Val accuracy: 0.6254375729288215
    Epoch: 4, Train loss: 0.5550544677018404, Train accuracy: 0.6850616931915283
    Val loss: 0.58931936110769, Val accuracy: 0.6429404900816803
    Epoch: 5, Train loss: 0.5597714431367743, Train accuracy: 0.6793931126594543
    Val loss: 0.568062335252762, Val accuracy: 0.6592765460910152
    Epoch: 6, Train loss: 0.5560962881553805, Train accuracy: 0.678892970085144
    Val loss: 0.5763750416891915, Val accuracy: 0.6581096849474912
    Epoch: 7, Train loss: 0.5481999519567722, Train accuracy: 0.6913971304893494
    Val loss: 0.5815095092569079, Val accuracy: 0.6522753792298717
    Epoch: 8, Train loss: 0.5429527659382809, Train accuracy: 0.6902300715446472
    Val loss: 0.618509488446372, Val accuracy: 0.6079346557759626
    Epoch: 9, Train loss: 0.5326030437709411, Train accuracy: 0.7029009461402893
    Val loss: 0.6136069340365273, Val accuracy: 0.6091015169194866
    Epoch: 10, Train loss: 0.5398539549193807, Train accuracy: 0.6997332572937012
    Val loss: 0.5764987745455333, Val accuracy: 0.6417736289381564
    Epoch: 11, Train loss: 0.5335546633170898, Train accuracy: 0.702400803565979
    Val loss: 0.5889025032520295, Val accuracy: 0.6511085180863477
    Early stopped training at epoch 11
```



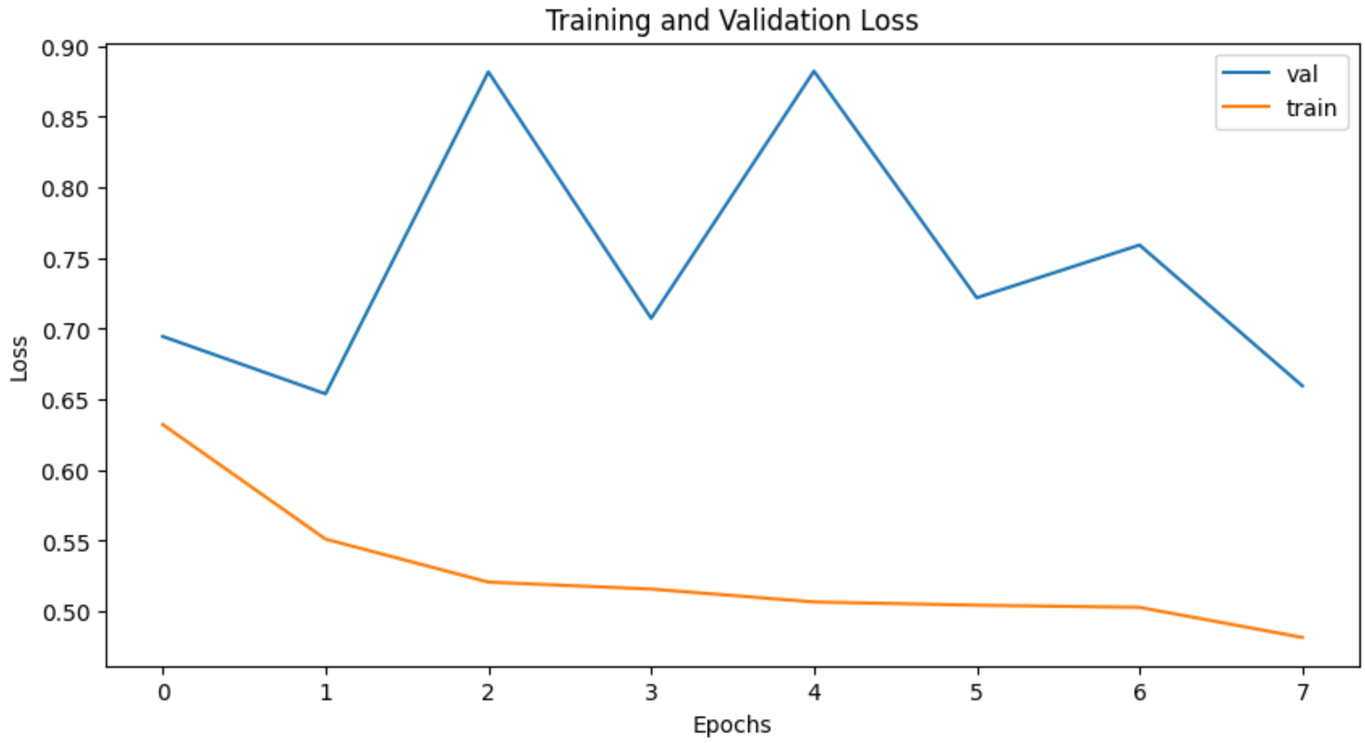


## batch size of 32, learning rate of 0.001

```
train_loader_3min = load_data(HypoDataset(ecg_3min_train, art_3min_train, eeg_3mi
val_loader_3min = load_data(HypoDataset(ecg_3min_val, art_3min_val, eeg_3min_val,
test_loader_3min = load_data(HypoDataset(ecg_3min_test, art_3min_test, eeg_3min_t

train_loader_5min = load_data(HypoDataset(ecg_5min_train, art_5min_train, eeg_5mi
val_loader_5min = load_data(HypoDataset(ecg_5min_val, art_5min_val, eeg_5min_val,
test_loader_5min = load_data(HypoDataset(ecg_5min_test, art_5min_test, eeg_5min_t

train_loader_10min = load_data(HypoDataset(ecg_10min_train, art_10min_train, eeg_
val_loader_10min = load_data(HypoDataset(ecg_10min_val, art_10min_val, eeg_10min_
test_loader_10min = load_data(HypoDataset(ecg_10min_test, art_10min_test, eeg_10m

train_loader_15min = load_data(HypoDataset(ecg_15min_train, art_15min_train, eeg_
val_loader_15min = load_data(HypoDataset(ecg_15min_val, art_15min_val, eeg_15min_
test_loader_15min = load_data(HypoDataset(ecg_15min_test, art_15min_test, eeg_15m


loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_batch_32_lr001.
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.5893906813990014, Train accuracy: 0.6952823996543884
Val loss: 0.5265198711838042, Val accuracy: 0.7280898876404495
Epoch: 1, Train loss: 0.49767022341757594, Train accuracy: 0.755455732345581
Val loss: 0.5221726096102169, Val accuracy: 0.7370786516853932
Epoch: 2, Train loss: 0.4640996441608522, Train accuracy: 0.7766367197036743
Val loss: 0.4980977858815874, Val accuracy: 0.755056179775281
Epoch: 3, Train loss: 0.4475786182096895, Train accuracy: 0.7907573580741882
Val loss: 0.592762015759945, Val accuracy: 0.7213483146067415
Epoch: 4, Train loss: 0.4224546831746769, Train accuracy: 0.8063222169876099
Val loss: 0.5026046229260308, Val accuracy: 0.7640449438202247
Epoch: 5, Train loss: 0.4140305474725706, Train accuracy: 0.8165917992591858
Val loss: 0.4470753446221351, Val accuracy: 0.7797752808988764
Epoch: 6, Train loss: 0.39070290950663744, Train accuracy: 0.8313543200492859
Val loss: 0.4389637910893984, Val accuracy: 0.8
Epoch: 7, Train loss: 0.3813048980661473, Train accuracy: 0.832317054271698
Val loss: 0.44120069061006817, Val accuracy: 0.7887640449438202
Epoch: 8, Train loss: 0.37008622529723983, Train accuracy: 0.8342426419258118
Val loss: 0.440295218357018, Val accuracy: 0.8044943820224719
Epoch: 9, Train loss: 0.35220551733716615, Train accuracy: 0.8440307974815369
Val loss: 0.44070154534918915, Val accuracy: 0.8067415730337079
Epoch: 10, Train loss: 0.3389106915835698, Train accuracy: 0.856867790222168
Val loss: 0.4407861541424479, Val accuracy: 0.8
Epoch: 11, Train loss: 0.3376954544875986, Train accuracy: 0.853498101234436
Val loss: 0.47639329731464397, Val accuracy: 0.8
Epoch: 12, Train loss: 0.3284002544408585, Train accuracy: 0.8578305244445801
Val loss: 0.44796438249094145, Val accuracy: 0.7943820224719101
Early stopped training at epoch 12
```

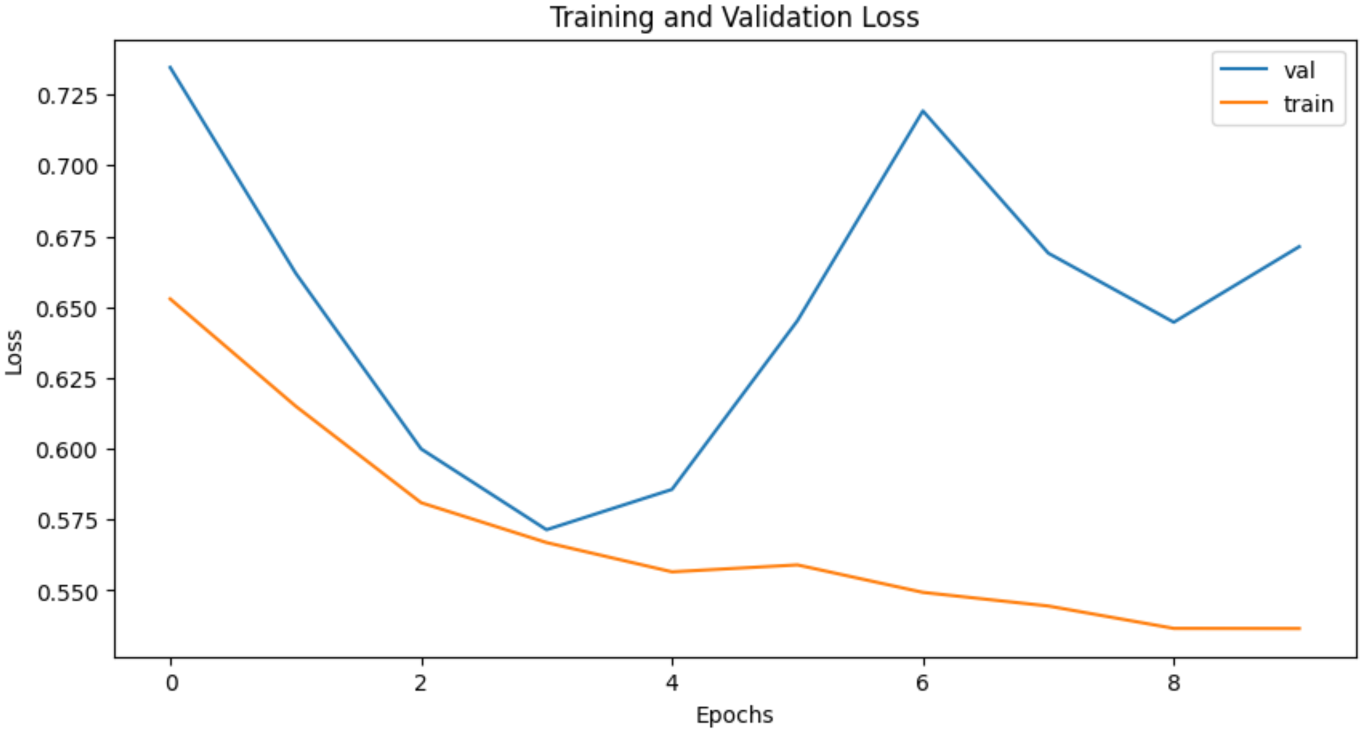


```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_batch_32_lr001.
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.6322526441722891, Train accuracy: 0.6436539888381958
Val loss: 0.6945784283535821, Val accuracy: 0.5565610859728507
Epoch: 1, Train loss: 0.5512032901411974, Train accuracy: 0.710590124130249
Val loss: 0.6539705959813934, Val accuracy: 0.6414027149321267
Epoch: 2, Train loss: 0.5207583422707085, Train accuracy: 0.7295068502426147
Val loss: 0.8817940418209348, Val accuracy: 0.5735294117647058
Epoch: 3, Train loss: 0.515845769080021, Train accuracy: 0.7282134294509888
Val loss: 0.7074199733989579, Val accuracy: 0.6221719457013575
Epoch: 4, Train loss: 0.5067253539036047, Train accuracy: 0.7320937514305115
Val loss: 0.882312923669815, Val accuracy: 0.5961538461538461
Epoch: 5, Train loss: 0.5044180508621682, Train accuracy: 0.7367825508117676
Val loss: 0.72196897651468, Val accuracy: 0.667420814479638
Epoch: 6, Train loss: 0.5028981796849035, Train accuracy: 0.7459983825683594
Val loss: 0.7592645319444793, Val accuracy: 0.6346153846153846
Epoch: 7, Train loss: 0.481587578480342, Train accuracy: 0.7502021193504333
Val loss: 0.6596198699304037, Val accuracy: 0.6289592760180995
Early stopped training at epoch 7
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5
```

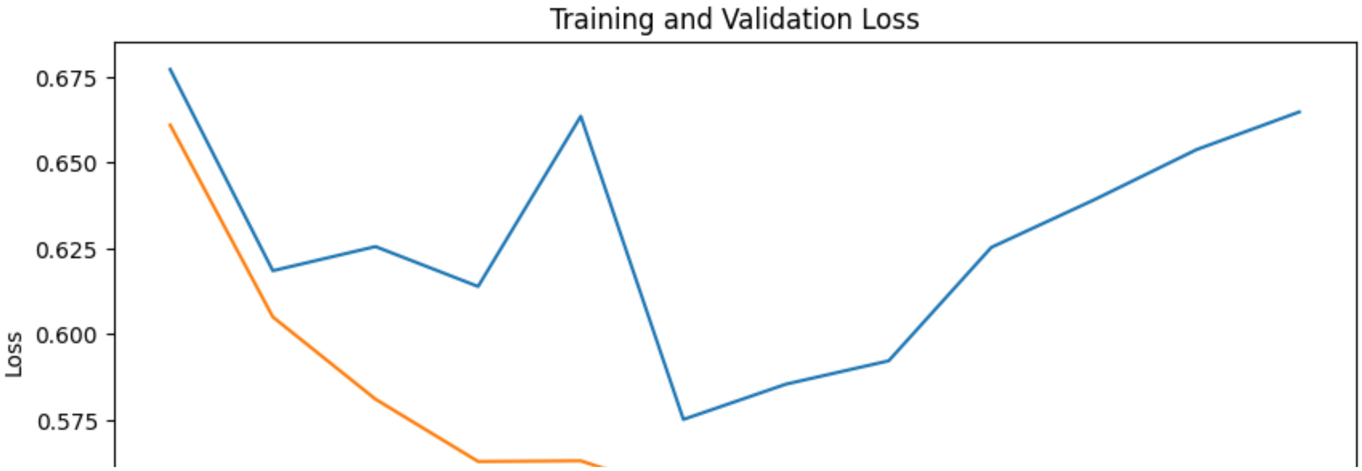
```
# 10 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_batch_32_lr001
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
Epoch: 0, Train loss: 0.652879525686648, Train accuracy: 0.6206270456314087
Val loss: 0.7345897172178542, Val accuracy: 0.4618937644341801
Epoch: 1, Train loss: 0.6150424218413854, Train accuracy: 0.6534653306007385
Val loss: 0.6619892567396164, Val accuracy: 0.5739030023094688
Epoch: 2, Train loss: 0.5809500876432991, Train accuracy: 0.6707921028137207
Val loss: 0.5999479655708584, Val accuracy: 0.6501154734411085
Epoch: 3, Train loss: 0.566884186558991, Train accuracy: 0.6884488463401794
Val loss: 0.5714094936847686, Val accuracy: 0.6570438799076213
Epoch: 4, Train loss: 0.5565515115709588, Train accuracy: 0.6863036155700684
Val loss: 0.5856445836169378, Val accuracy: 0.6189376443418014
Epoch: 5, Train loss: 0.5589985322244096, Train accuracy: 0.685973584651947
Val loss: 0.6452010380370278, Val accuracy: 0.6085450346420324
Epoch: 6, Train loss: 0.5493023123088057, Train accuracy: 0.6902640461921692
Val loss: 0.7192337342670986, Val accuracy: 0.5496535796766744
Epoch: 7, Train loss: 0.5444888821136047, Train accuracy: 0.6975247263908386
Val loss: 0.6690150180033275, Val accuracy: 0.5739030023094688
Epoch: 8, Train loss: 0.5365920784449814, Train accuracy: 0.7033003568649292
Val loss: 0.6446535566023417, Val accuracy: 0.5854503464203233
Epoch: 9, Train loss: 0.536543860175822, Train accuracy: 0.7033003568649292
Val loss: 0.6713319686906679, Val accuracy: 0.5577367205542725
Early stopped training at epoch 9
```

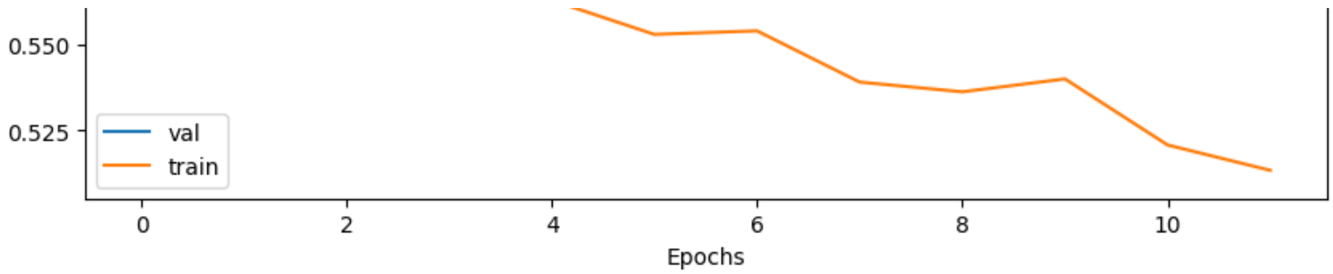


```
loss_func = nn.BCEWithLogitsLoss()
```

```
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = my_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_batch_32_lr001
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
```

```
    Epoch: 0, Train loss: 0.6608558075155326, Train accuracy: 0.611037015914917
    Val loss: 0.6770786797558818, Val accuracy: 0.5624270711785297
    Epoch: 1, Train loss: 0.6048799892710462, Train accuracy: 0.6512170433998108
    Val loss: 0.6183609543023285, Val accuracy: 0.6009334889148191
    Epoch: 2, Train loss: 0.5809101047377541, Train accuracy: 0.6798933148384094
    Val loss: 0.6254052188661362, Val accuracy: 0.5857642940490082
    Epoch: 3, Train loss: 0.5627291405268533, Train accuracy: 0.690396785736084
    Val loss: 0.6137712586809086, Val accuracy: 0.6102683780630105
    Epoch: 4, Train loss: 0.562925008246405, Train accuracy: 0.6868956089019775
    Val loss: 0.6634174519115024, Val accuracy: 0.5717619603267211
    Epoch: 5, Train loss: 0.5530034826969695, Train accuracy: 0.6888962984085083
    Val loss: 0.5749904921761264, Val accuracy: 0.6569428238039673
    Epoch: 6, Train loss: 0.5540292856791847, Train accuracy: 0.6958986520767212
    Val loss: 0.5852854406392132, Val accuracy: 0.6499416569428238
    Epoch: 7, Train loss: 0.5391159465945773, Train accuracy: 0.6988996267318726
    Val loss: 0.5920857047593152, Val accuracy: 0.6546091015169195
    Epoch: 8, Train loss: 0.5362823654827017, Train accuracy: 0.7052350640296936
    Val loss: 0.6251722949522512, Val accuracy: 0.6254375729288215
    Epoch: 9, Train loss: 0.5400372954876115, Train accuracy: 0.6983994841575623
    Val loss: 0.6390254939043963, Val accuracy: 0.5845974329054843
    Epoch: 10, Train loss: 0.5207999052704076, Train accuracy: 0.714404821395874
    Val loss: 0.6536952588293289, Val accuracy: 0.5717619603267211
    Epoch: 11, Train loss: 0.5134242811414471, Train accuracy: 0.725908637046814
    Val loss: 0.6646714762405114, Val accuracy: 0.5122520420070011
    Early stopped training at epoch 11
```





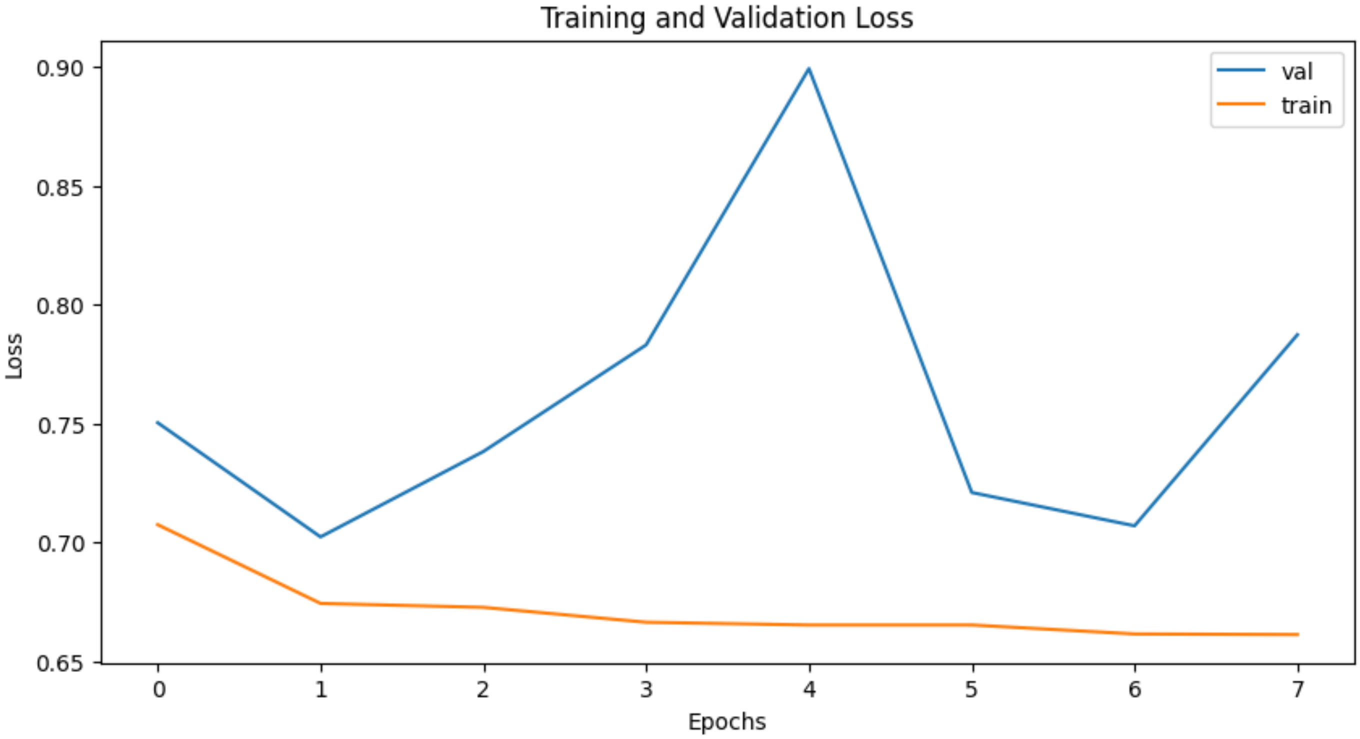
## Ablations

Given the best performance with a batch size of 32 and a lr of 0.001 with the current dataset, testing of ablations with these settings:

```
train_loader_3min = load_data(HypoDataset(ecg_3min_train, art_3min_train, eeg_3mi
val_loader_3min = load_data(HypoDataset(ecg_3min_val, art_3min_val, eeg_3min_val,
test_loader_3min = load_data(HypoDataset(ecg_3min_test, art_3min_test, eeg_3min_t

train_loader_5min = load_data(HypoDataset(ecg_5min_train, art_5min_train, eeg_5mi
val_loader_5min = load_data(HypoDataset(ecg_5min_val, art_5min_val, eeg_5min_val,
test_loader_5min = load_data(HypoDataset(ecg_5min_test, art_5min_test, eeg_5min_t

train_loader_10min = load_data(HypoDataset(ecg_10min_train, art_10min_train, eeg_
val_loader_10min = load_data(HypoDataset(ecg_10min_val, art_10min_val, eeg_10min_
test_loader_10min = load_data(HypoDataset(ecg_10min_test, art_10min_test, eeg_10m

train_loader_15min = load_data(HypoDataset(ecg_15min_train, art_15min_train, eeg_
val_loader_15min = load_data(HypoDataset(ecg_15min_val, art_15min_val, eeg_15min_
test_loader_15min = load_data(HypoDataset(ecg_15min_test, art_15min_test, eeg_15m
```

### Test model limited to ECG data

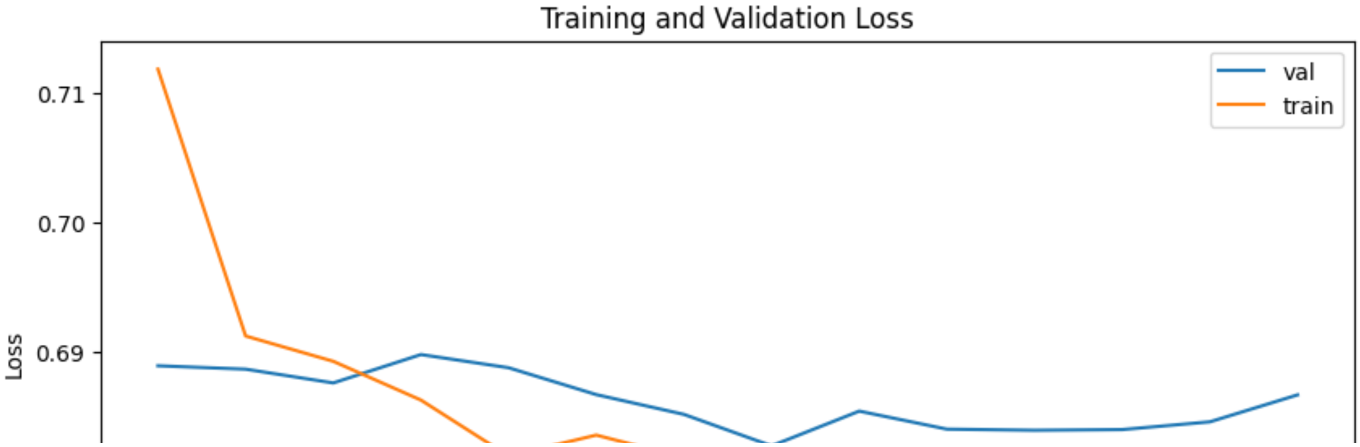
```
class ecg_model(nn.Module):
  # use this class to define your model
  def __init__(self):
    super().__init__()

    self.encoder_ecg = nn.Sequential(
                        nn.Conv1d(in_channels=1, out_channels=1, kernel_size=3,st
                        )

    self.layer1_ecg = nn.Sequential(
                        nn.BatchNorm1d(1),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(1,2,15,stride=1, padding='same'),
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Conv1d(2,2,15,stride=1, padding='same')
                        )

    self.maxpool_ecg_layer1 = nn.MaxPool1d(2, 2)

    self.layer2_ecg = nn.Sequential(
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(2,2,15,stride=1, padding='same'),
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Conv1d(2,2,15,stride=1, padding='same')
                        )

    self.layer3_ecg = nn.Sequential(
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(2,2,15,stride=1, padding='same'),
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Conv1d(2,2,15,stride=1, padding='same')
                        )

    self.maxpool_ecg_layer3 = nn.MaxPool1d(2, 2)

    self.layer4_ecg = nn.Sequential(
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(2,2,15,stride=1, padding='same'),
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Conv1d(2,2,15,stride=1, padding='same')
                        )

    self.layer5_ecg = nn.Sequential(
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(2,2,15,stride=1, padding='same'),
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Conv1d(2,2,15,stride=1, padding='same')
                        )

    self.maxpool_ecg_layer5 = nn.MaxPool1d(2, 2)

    self.layer6_ecg = nn.Sequential(
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(2,4,15,stride=1, padding='same'),
                        nn.BatchNorm1d(4),
                        nn.ReLU(),
                        nn.Conv1d(4,4,15,stride=1, padding='same')
                        )

    self.layer7_ecg = nn.Sequential(
                        nn.BatchNorm1d(4),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(4,4,7,stride=1, padding='same'),
                        nn.BatchNorm1d(4),
                        nn.ReLU(),
                        nn.Conv1d(4,4,7,stride=1, padding='same')
                        )

    self.maxpool_ecg_layer7 = nn.MaxPool1d(2, 2)

    self.layer8_ecg = nn.Sequential(
                        nn.BatchNorm1d(4),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(4,4,7,stride=1, padding='same'),
                        nn.BatchNorm1d(4),
                        nn.ReLU(),
                        nn.Conv1d(4,4,7,stride=1, padding='same')
                        )

    self.layer9_ecg = nn.Sequential(
                        nn.BatchNorm1d(4),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(4,4,7,stride=1, padding='same'),
                        nn.BatchNorm1d(4),
                        nn.ReLU(),
                        nn.Conv1d(4,4,7,stride=1, padding='same')
                        )

    self.maxpool_ecg_layer9 = nn.MaxPool1d(2, 2)

    self.layer10_ecg = nn.Sequential(
                        nn.BatchNorm1d(4),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(4,6,7,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Conv1d(6,6,7,stride=1, padding='same')
                        )

    self.layer11_ecg = nn.Sequential(
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(6,6,7,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Conv1d(6,6,7,stride=1, padding='same')
                        )

    self.maxpool_ecg_layer11 = nn.MaxPool1d(2, 2)

    self.layer12_ecg = nn.Sequential(
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(6,6,7,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        )

    self.linear_ecg = nn.Linear(468*6, 96)

    self.linear_combined1 = nn.Linear(96, 16)
    self.linear_combined2 = nn.Linear(16, 1)

    self.apply(self._init_weights)

  def _init_weights(self, module):

      if isinstance(module, nn.Conv1d):
        nn.init.kaiming_normal_(module.weight, nonlinearity='relu')

        if module.bias is not None:
          nn.init.zeros_(module.bias)

      elif isinstance(module, nn.BatchNorm1d):
        nn.init.constant_(module.weight, 1)
        nn.init.constant_(module.bias, 0)

      elif isinstance(module, nn.Linear):
        nn.init.xavier_normal_(module.weight)

        if module.bias is not None:
          nn.init.zeros_(module.bias)

  def forward(self, ecg, art, eeg):

    ecg = self.encoder_ecg(ecg)
    tmp = self.layer1_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer1(ecg)
    tmp = self.layer2_ecg(ecg)
    ecg = tmp + ecg
    tmp = self.layer3_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer3(ecg)
    tmp = self.layer4_ecg(ecg)
    ecg = tmp + ecg
    tmp = self.layer5_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer5(ecg)
    ecg = self.layer6_ecg(ecg)
    tmp = self.layer7_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer7(ecg)
    tmp = self.layer8_ecg(ecg)
    ecg = tmp + ecg
    tmp = self.layer9_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer9(ecg)
    ecg = self.layer10_ecg(ecg)
    tmp = self.layer11_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer11(ecg)
    tmp = self.layer12_ecg(ecg)
    ecg = tmp + ecg
    ecg = torch.flatten(ecg, 1)
    ecg = F.relu(self.linear_ecg(ecg))

    combined = F.relu(self.linear_combined1(ecg))
    logits = self.linear_combined2(combined)
    # probs = F.sigmoid(logits)

    return logits


loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = ecg_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_ecg.pth"
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.7074016989884235, Train accuracy: 0.5638639330863953
Val loss: 0.7502975527729308, Val accuracy: 0.41460674157303373
Epoch: 1, Train loss: 0.6742828913004316, Train accuracy: 0.5935494303703308
Val loss: 0.7022645452192853, Val accuracy: 0.4157303370786517
Epoch: 2, Train loss: 0.672629947916381, Train accuracy: 0.589698314666748
Val loss: 0.7381893247365953, Val accuracy: 0.4134831460674157
Epoch: 3, Train loss: 0.6663610928324895, Train accuracy: 0.5986841917037964
Val loss: 0.7830507712704795, Val accuracy: 0.4134831460674157
Epoch: 4, Train loss: 0.6652121972364394, Train accuracy: 0.6022143959999084
Val loss: 0.8992813676595691, Val accuracy: 0.4134831460674157
Epoch: 5, Train loss: 0.6652129086664613, Train accuracy: 0.6051027178764343
Val loss: 0.7209473699331282, Val accuracy: 0.4157303370786517
Epoch: 6, Train loss: 0.6614353379021867, Train accuracy: 0.6015725135803223
Val loss: 0.7069305756262371, Val accuracy: 0.4258426966292135
Epoch: 7, Train loss: 0.6611886906685052, Train accuracy: 0.605423629283905
Val loss: 0.7873196388993944, Val accuracy: 0.4157303370786517
Early stopped training at epoch 7
```

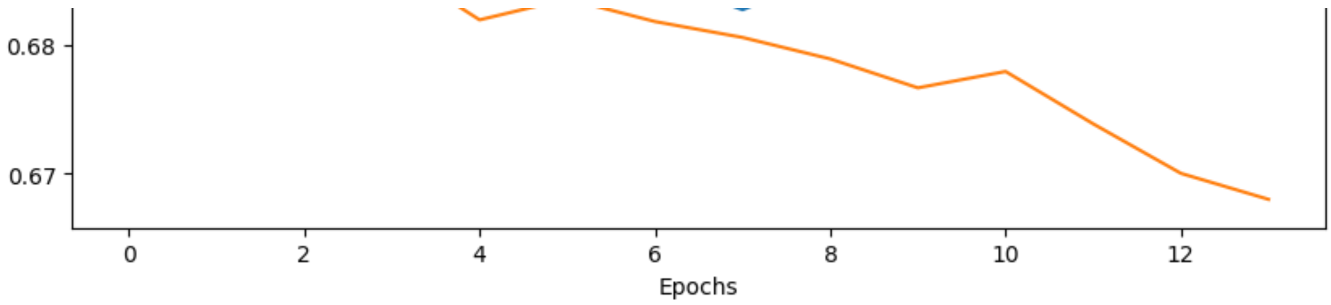


```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5
```

```
# 5 min model
torch.manual_seed(seed)
model = ecg_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_ecg.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
```

```
Epoch: 0, Train loss: 0.7118454230920673, Train accuracy: 0.5429264307022095
Val loss: 0.6888829512255532, Val accuracy: 0.5859728506787331
Epoch: 1, Train loss: 0.691187599857873, Train accuracy: 0.5503637790679932
Val loss: 0.6886200372661863, Val accuracy: 0.5859728506787331
Epoch: 2, Train loss: 0.6892275701247595, Train accuracy: 0.5561842918395996
Val loss: 0.6875534504652024, Val accuracy: 0.5859728506787331
Epoch: 3, Train loss: 0.6862280840341758, Train accuracy: 0.5578011274337769
Val loss: 0.6897510247571127, Val accuracy: 0.5859728506787331
Epoch: 4, Train loss: 0.6818973166188985, Train accuracy: 0.5600646734237671
Val loss: 0.6887305144752776, Val accuracy: 0.5859728506787331
Epoch: 5, Train loss: 0.6835173235367457, Train accuracy: 0.5603880286216736
Val loss: 0.6866507253476551, Val accuracy: 0.5859728506787331
Epoch: 6, Train loss: 0.6817672352686753, Train accuracy: 0.5602263808250427
Val loss: 0.6851196118763514, Val accuracy: 0.5859728506787331
Epoch: 7, Train loss: 0.6805362703343409, Train accuracy: 0.5628132820129395
Val loss: 0.6826829676117215, Val accuracy: 0.5859728506787331
Epoch: 8, Train loss: 0.678876847852693, Train accuracy: 0.5641067028045654
Val loss: 0.6853723930461065, Val accuracy: 0.584841628959276
Epoch: 9, Train loss: 0.6766417266769378, Train accuracy: 0.5660468935966492
Val loss: 0.683976743902479, Val accuracy: 0.5871040723981901
Epoch: 10, Train loss: 0.6779071637865219, Train accuracy: 0.5639450550079346
Val loss: 0.6838911324739457, Val accuracy: 0.584841628959276
Epoch: 11, Train loss: 0.6738548831392317, Train accuracy: 0.5675020217895508
Val loss: 0.6839436973844256, Val accuracy: 0.584841628959276
Epoch: 12, Train loss: 0.6700303273906415, Train accuracy: 0.574616014957428
Val loss: 0.6845437543732781, Val accuracy: 0.581447963800905
Epoch: 13, Train loss: 0.6680185776712824, Train accuracy: 0.5801131725311279
Val loss: 0.6866312005690165, Val accuracy: 0.584841628959276
Early stopped training at epoch 13
```

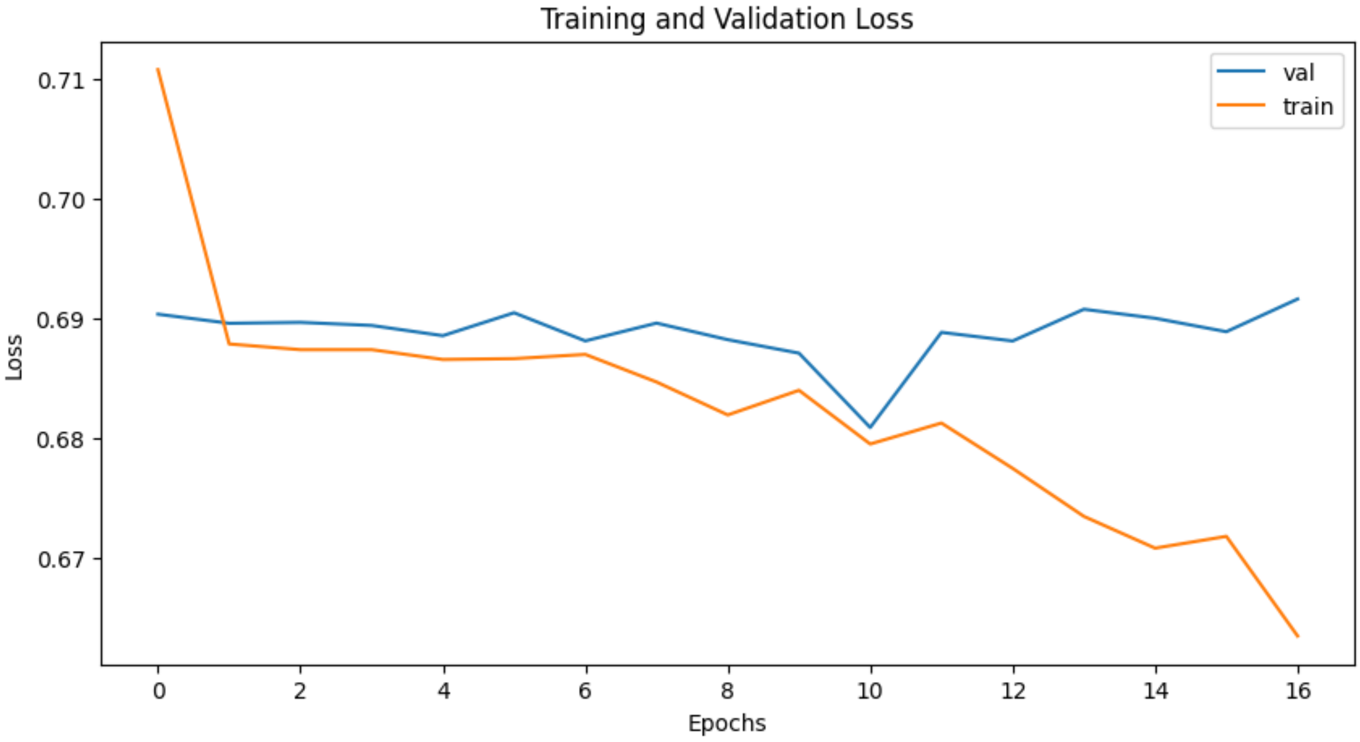




```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 10 min model
torch.manual_seed(seed)
model = ecg_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_ecg.pth"
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
```

```
Epoch: 0, Train loss: 0.7108280604035154, Train accuracy: 0.5412541031837463
Val loss: 0.6903990869011198, Val accuracy: 0.5739030023094688
Epoch: 1, Train loss: 0.6879122333951516, Train accuracy: 0.5526402592658997
Val loss: 0.6896400877407619, Val accuracy: 0.5739030023094688
Epoch: 2, Train loss: 0.6874361653532526, Train accuracy: 0.5518151521682739
Val loss: 0.6897169947624207, Val accuracy: 0.5739030023094688
Epoch: 3, Train loss: 0.6874363577798648, Train accuracy: 0.5514851212501526
Val loss: 0.6894584745168686, Val accuracy: 0.5739030023094688
Epoch: 4, Train loss: 0.68661603451562, Train accuracy: 0.553135335445404
Val loss: 0.6886035501956941, Val accuracy: 0.5739030023094688
Epoch: 5, Train loss: 0.6866853808019028, Train accuracy: 0.553135335445404
Val loss: 0.6905178917305812, Val accuracy: 0.5739030023094688
Epoch: 6, Train loss: 0.6870394601286834, Train accuracy: 0.5518151521682739
Val loss: 0.6881668312208993, Val accuracy: 0.5739030023094688
Epoch: 7, Train loss: 0.6847354716987106, Train accuracy: 0.5537953972816467
Val loss: 0.6896543779543469, Val accuracy: 0.5450346420323325
Epoch: 8, Train loss: 0.6819824429628479, Train accuracy: 0.555940568447113
Val loss: 0.6882715054920743, Val accuracy: 0.5727482678983834
Epoch: 9, Train loss: 0.6840404803603396, Train accuracy: 0.5567656755447388
Val loss: 0.6871452778577806, Val accuracy: 0.5715935334872979
Epoch: 10, Train loss: 0.6795552256477154, Train accuracy: 0.5589109063148499
Val loss: 0.6809344568422862, Val accuracy: 0.5762124711316398
```

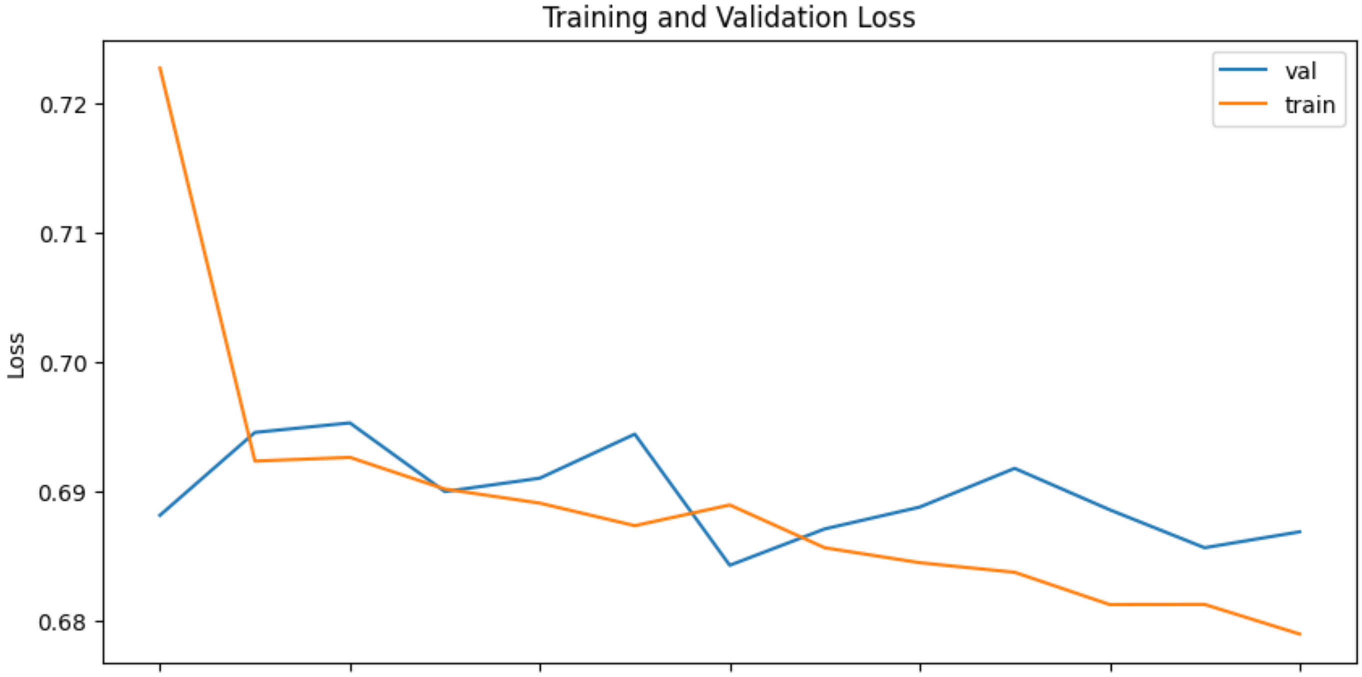
```
Epoch: 11, Train loss: 0.6813052658594088, Train accuracy: 0.5599009990692139
Val loss: 0.6888811013528279, Val accuracy: 0.5739030023094688
Epoch: 12, Train loss: 0.677520225897874, Train accuracy: 0.5608910918235779
Val loss: 0.6881615306649891, Val accuracy: 0.5727482678983834
Epoch: 13, Train loss: 0.6735089041218899, Train accuracy: 0.5651814937591553
Val loss: 0.6908118618386132, Val accuracy: 0.5681293302540416
Epoch: 14, Train loss: 0.6708544934543446, Train accuracy: 0.563036322593689
Val loss: 0.6900598555803299, Val accuracy: 0.5715935334872979
Epoch: 15, Train loss: 0.6718400081785598, Train accuracy: 0.5694719552993774
Val loss: 0.688936620950699, Val accuracy: 0.5727482678983834
Epoch: 16, Train loss: 0.6635191858798364, Train accuracy: 0.5673267245292664
Val loss: 0.6916650959423611, Val accuracy: 0.5669745958429562
Early stopped training at epoch 16
```

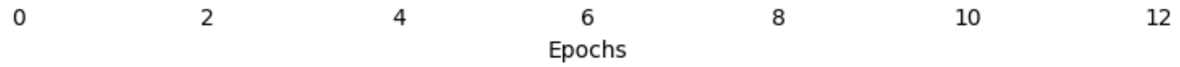


```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = ecg_model()
```

```
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_ecg.pth"
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
```

```
Epoch: 0, Train loss: 0.7227167989580422, Train accuracy: 0.5231744050979614
Val loss: 0.6881333744084392, Val accuracy: 0.5717619603267211
Epoch: 1, Train loss: 0.6923282812062245, Train accuracy: 0.5338445901870728
Val loss: 0.6945432225863138, Val accuracy: 0.4340723453908985
Epoch: 2, Train loss: 0.6926127703001437, Train accuracy: 0.545681893825531
Val loss: 0.695279582783028, Val accuracy: 0.4329054842473746
Epoch: 3, Train loss: 0.6901641402414697, Train accuracy: 0.551350474357605
Val loss: 0.6899687647819519, Val accuracy: 0.5659276546091015
Epoch: 4, Train loss: 0.6890739807688264, Train accuracy: 0.5478492975234985
Val loss: 0.6910050555511756, Val accuracy: 0.5612602100350058
Epoch: 5, Train loss: 0.6873321926526206, Train accuracy: 0.5493497848510742
Val loss: 0.6944110658433702, Val accuracy: 0.43173862310385064
Epoch: 6, Train loss: 0.6889337655622667, Train accuracy: 0.5451817512512207
Val loss: 0.6842694326683327, Val accuracy: 0.5694282380396732
Epoch: 7, Train loss: 0.6856246116996885, Train accuracy: 0.5508502721786499
Val loss: 0.6870804208296317, Val accuracy: 0.5694282380396732
Epoch: 8, Train loss: 0.6844687773291768, Train accuracy: 0.5531843900680542
Val loss: 0.6887689718493709, Val accuracy: 0.5694282380396732
Epoch: 9, Train loss: 0.6837268875972078, Train accuracy: 0.5546848773956299
Val loss: 0.6917637851503161, Val accuracy: 0.5705950991831972
Epoch: 10, Train loss: 0.6812318786655756, Train accuracy: 0.553851306438446
Val loss: 0.6885625830403082, Val accuracy: 0.5694282380396732
Epoch: 11, Train loss: 0.6812419069731541, Train accuracy: 0.5636879205703735
Val loss: 0.6856255619614212, Val accuracy: 0.5705950991831972
Epoch: 12, Train loss: 0.6789521890984014, Train accuracy: 0.5693564414978027
Val loss: 0.6868566075960795, Val accuracy: 0.5647607934655776
Early stopped training at epoch 12
```



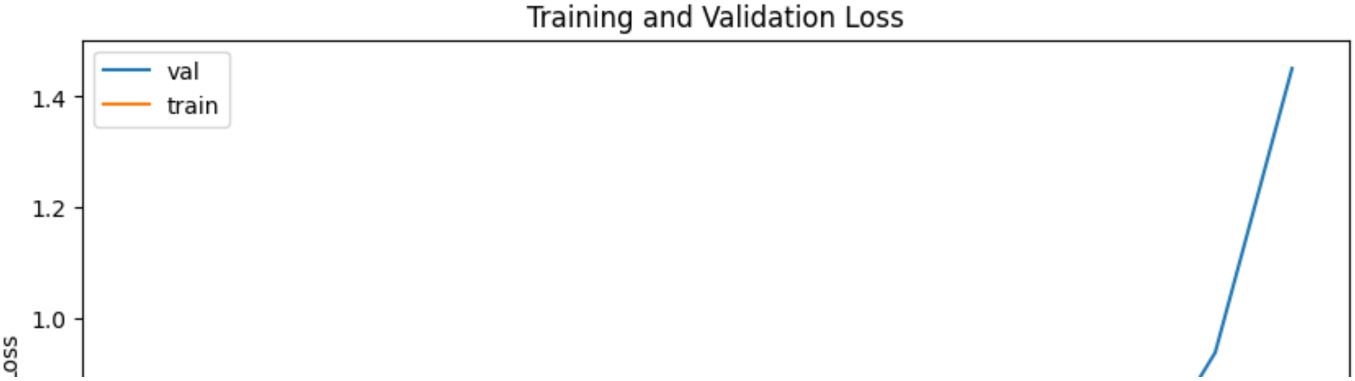


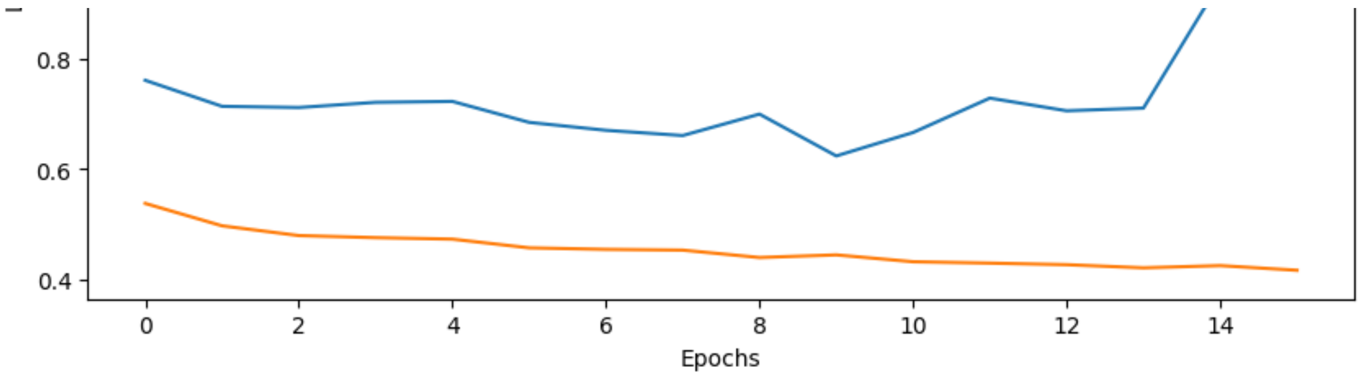
## Test model limited to ART (blood pressure) data

```
class art_model(nn.Module):
  # use this class to define your model
  def __init__(self):

    super().__init__()

    self.encoder_art = nn.Sequential(
                    nn.Conv1d(1,1,3,stride=1, padding='same')
                    )

    self.layer1_art = nn.Sequential(
                    nn.BatchNorm1d(1),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(1,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.maxpool_art_layer1 = nn.MaxPool1d(2, 2)

    self.layer2_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.layer3_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.maxpool_art_layer3 = nn.MaxPool1d(2, 2)

    self.layer4_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.layer5_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.maxpool_art_layer5 = nn.MaxPool1d(2, 2)

    self.layer6_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,4,15,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Conv1d(4,4,15,stride=1, padding='same')
                    )

    self.layer7_art = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

    self.maxpool_art_layer7 = nn.MaxPool1d(2, 2)

    self.layer8_art = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

    self.layer9_art = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

    self.maxpool_art_layer9 = nn.MaxPool1d(2, 2)

    self.layer10_art = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,6,7,stride=1, padding='same'),
                    nn.BatchNorm1d(6),
                    nn.ReLU(),
                    nn.Conv1d(6,6,7,stride=1, padding='same')
                    )

    self.layer11_art = nn.Sequential(
                    nn.BatchNorm1d(6),
                    nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(6,6,7,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Conv1d(6,6,7,stride=1, padding='same')
                        )

    self.maxpool_art_layer11 = nn.MaxPool1d(2, 2)

    self.layer12_art = nn.Sequential(
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(6,6,7,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        )

    self.linear_art = nn.Linear(468*6, 96)

    self.linear_combined1 = nn.Linear(96, 16)
    self.linear_combined2 = nn.Linear(16, 1)

    self.apply(self._init_weights)

  def _init_weights(self, module):

      if isinstance(module, nn.Conv1d):
        nn.init.kaiming_normal_(module.weight, nonlinearity='relu')

        if module.bias is not None:
          nn.init.zeros_(module.bias)

      elif isinstance(module, nn.BatchNorm1d):
        nn.init.constant_(module.weight, 1)
        nn.init.constant_(module.bias, 0)

      elif isinstance(module, nn.Linear):
        nn.init.xavier_normal_(module.weight)

        if module.bias is not None:
          nn.init.zeros_(module.bias)

  def forward(self, ecg, art, eeg):
    art = self.encoder_art(art)
    tmp = self.layer1_art(art)
    art = tmp + art
    art = self.maxpool_art_layer1(art)
    tmp = self.layer2_art(art)
    art = tmp + art
    tmp = self.layer3_art(art)
    art = tmp + art
    art = self.maxpool_art_layer3(art)
    tmp = self.layer4_art(art)
    art = tmp + art
    tmp = self.layer5_art(art)
    art = tmp + art
    art = self.maxpool_art_layer5(art)
    art = self.layer6_art(art)
    tmp = self.layer7_art(art)
    art = tmp + art
    art = self.maxpool_art_layer7(art)
    tmp = self.layer8_art(art)
    art = tmp + art
    tmp = self.layer9_art(art)
    art = tmp + art
    art = self.maxpool_art_layer9(art)
    art = self.layer10_art(art)
    tmp = self.layer11_art(art)
    art = tmp + art
    art = self.maxpool_art_layer11(art)
    tmp = self.layer12_art(art)
    art = tmp + art
    art = torch.flatten(art, 1)
    art = F.relu(self.linear_art(art))

    combined = F.relu(self.linear_combined1(art))
    logits = self.linear_combined2(combined)
    # probs = F.sigmoid(logits)

    return logits


loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
```

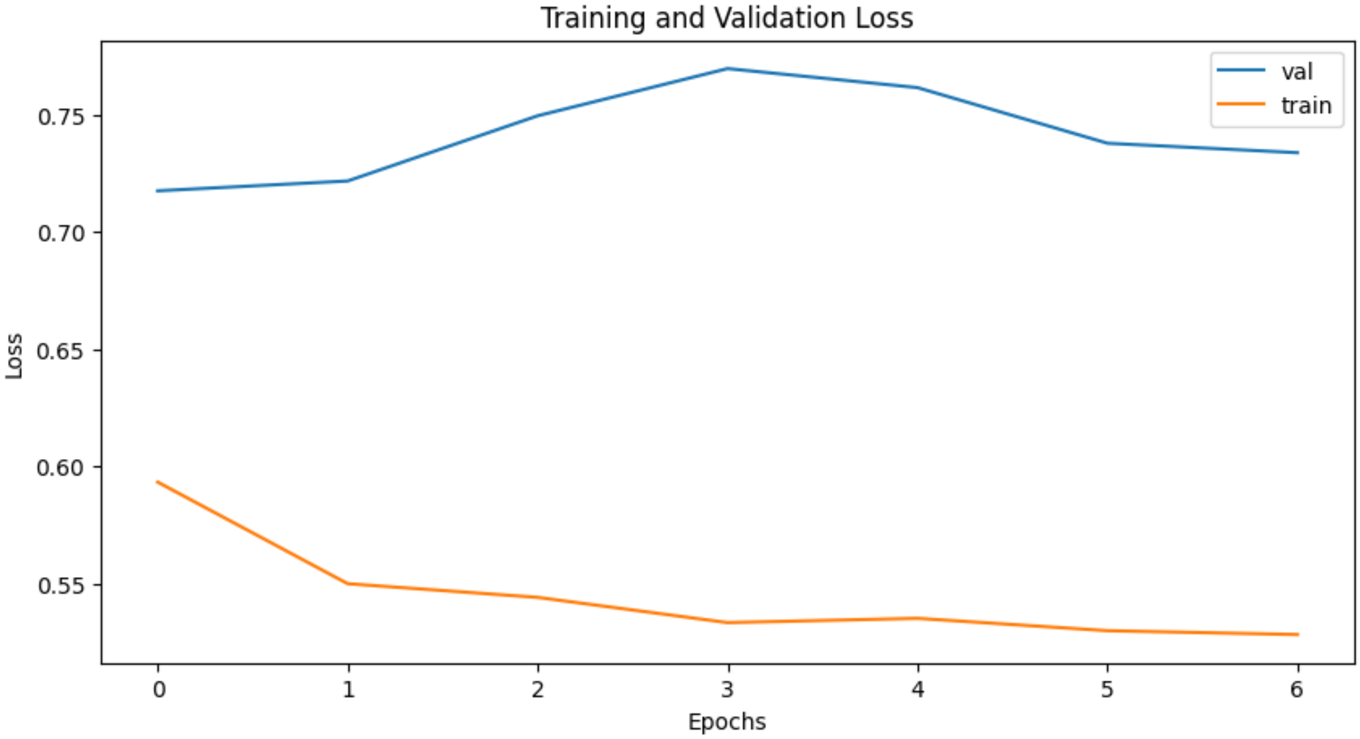
```
torch.manual_seed(seed)
model = art_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_art.pth"
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
```

```
Epoch: 0, Train loss: 0.5384343750822682, Train accuracy: 0.7259306907653809
Val loss: 0.7606342583894729, Val accuracy: 0.41460674157303373
Epoch: 1, Train loss: 0.4980108188955407, Train accuracy: 0.7467907667160034
Val loss: 0.7136889334235872, Val accuracy: 0.4157303370786517
Epoch: 2, Train loss: 0.48040635212395094, Train accuracy: 0.7541720271110535
Val loss: 0.7115428490298135, Val accuracy: 0.3089887640449438
Epoch: 3, Train loss: 0.4766531684585346, Train accuracy: 0.7586649656295776
Val loss: 0.7208844444581441, Val accuracy: 0.4191011235955056
Epoch: 4, Train loss: 0.4739765469008753, Train accuracy: 0.7633183598518372
Val loss: 0.72254424648625, Val accuracy: 0.5337078651685393
Epoch: 5, Train loss: 0.45825584892743054, Train accuracy: 0.7703787088394165
Val loss: 0.6847305787461143, Val accuracy: 0.5831460674157304
Epoch: 6, Train loss: 0.45548856354035216, Train accuracy: 0.7719833254814148
Val loss: 0.6704048216342926, Val accuracy: 0.5764044943820225
Epoch: 7, Train loss: 0.45416146585662803, Train accuracy: 0.7727856040000916
Val loss: 0.6609617109809603, Val accuracy: 0.6494382022471911
Epoch: 8, Train loss: 0.4407815148190754, Train accuracy: 0.7844993472099304
Val loss: 0.6996221478496277, Val accuracy: 0.5685393258426966
Epoch: 9, Train loss: 0.4455486523783712, Train accuracy: 0.7772785425186157
Val loss: 0.6241342191185271, Val accuracy: 0.7056179775280899
Epoch: 10, Train loss: 0.4331554222627231, Train accuracy: 0.7819319367408752
Val loss: 0.6664293546761785, Val accuracy: 0.6876404494382022
Epoch: 11, Train loss: 0.43071605427733434, Train accuracy: 0.784017980098724
Val loss: 0.7285249999591283, Val accuracy: 0.6640449438202247
Epoch: 12, Train loss: 0.4278643011893785, Train accuracy: 0.7862644195556641
Val loss: 0.7055945651871817, Val accuracy: 0.6382022471910113
Epoch: 13, Train loss: 0.42216980093862955, Train accuracy: 0.788831830024719
Val loss: 0.710450906838689 5, Val accuracy: 0.5932584269662922
Epoch: 14, Train loss: 0.42617979917354853, Train accuracy: 0.791880607604980
Val loss: 0.9380584997790201, Val accuracy: 0.5955056179775281
Epoch: 15, Train loss: 0.41779375080119663, Train accuracy: 0.802471101284027
Val loss: 1.4506771096161435, Val accuracy: 0.49887640449438203
Early stopped training at epoch 15
```





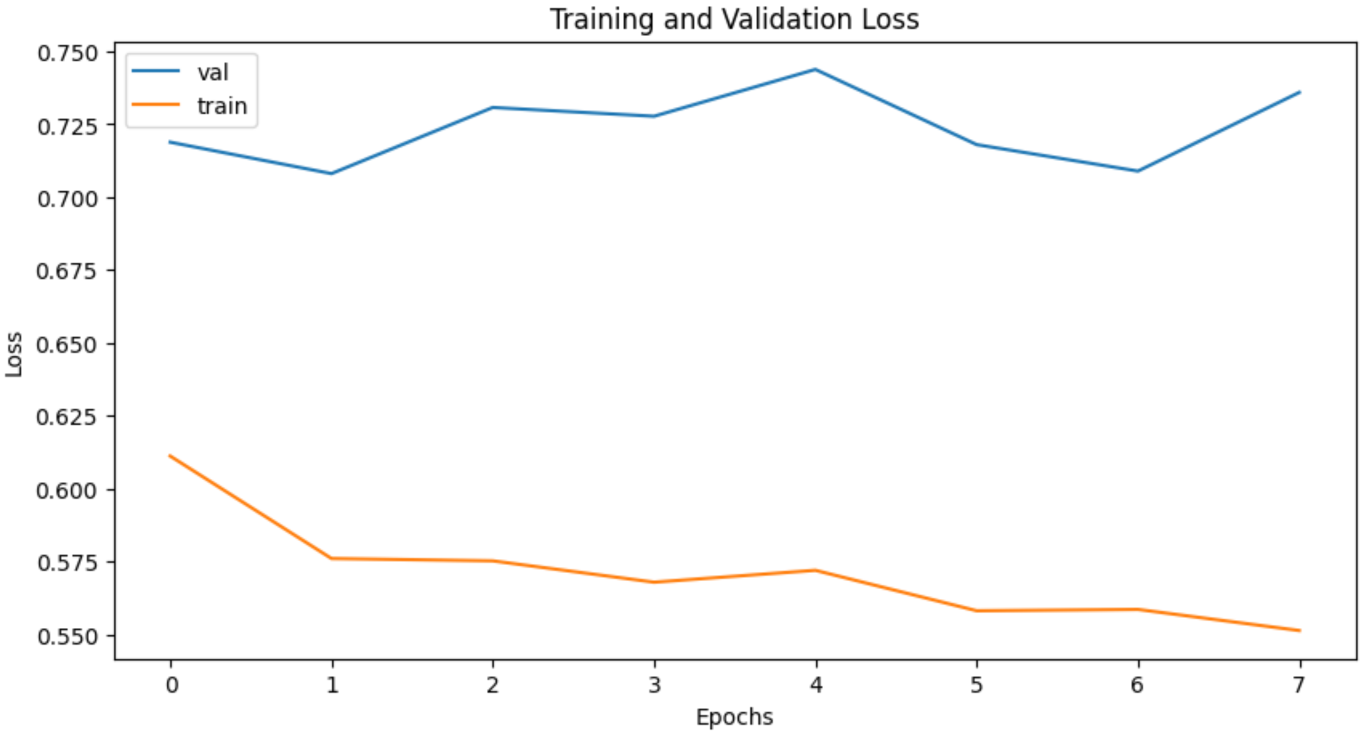
```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = art_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_art.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.5932531462625428, Train accuracy: 0.6789005398750305
Val loss: 0.7175564510481697, Val accuracy: 0.415158371040724
Epoch: 1, Train loss: 0.5499022249069152, Train accuracy: 0.7080032229423523
Val loss: 0.7217519858053753, Val accuracy: 0.28054298642533937
Epoch: 2, Train loss: 0.5440642818772417, Train accuracy: 0.7139854431152344
Val loss: 0.7495728326695307, Val accuracy: 0.415158371040724
Epoch: 3, Train loss: 0.5332933126964584, Train accuracy: 0.7126920223236084
Val loss: 0.7697183894259589, Val accuracy: 0.41402714932126694
Epoch: 4, Train loss: 0.535117141291704, Train accuracy: 0.7073565125465393
Val loss: 0.761596313544682, Val accuracy: 0.415158371040724
Epoch: 5, Train loss: 0.5298442129946179, Train accuracy: 0.7173807621002197
Val loss: 0.7378618163721902, Val accuracy: 0.415158371040724
Epoch: 6, Train loss: 0.5282027476144203, Train accuracy: 0.7178658246994019
Val loss: 0.733835133058684, Val accuracy: 0.415158371040724
Early stopped training at epoch 6
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 10 min model
```

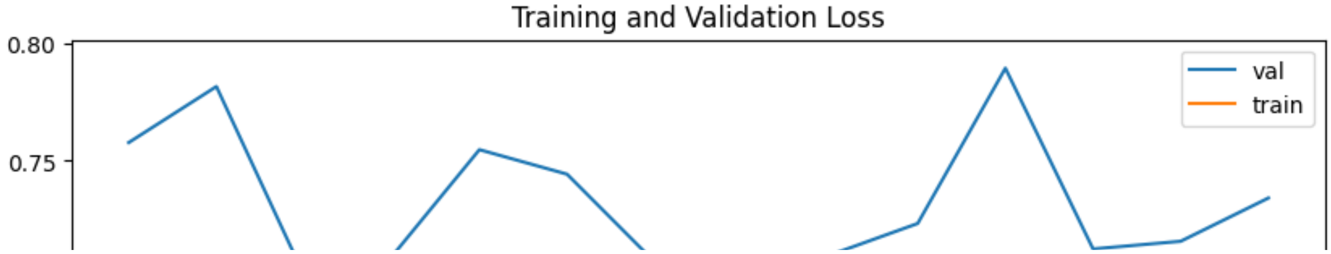
```
torch.manual_seed(seed)
model = art_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_art.pth"
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
```

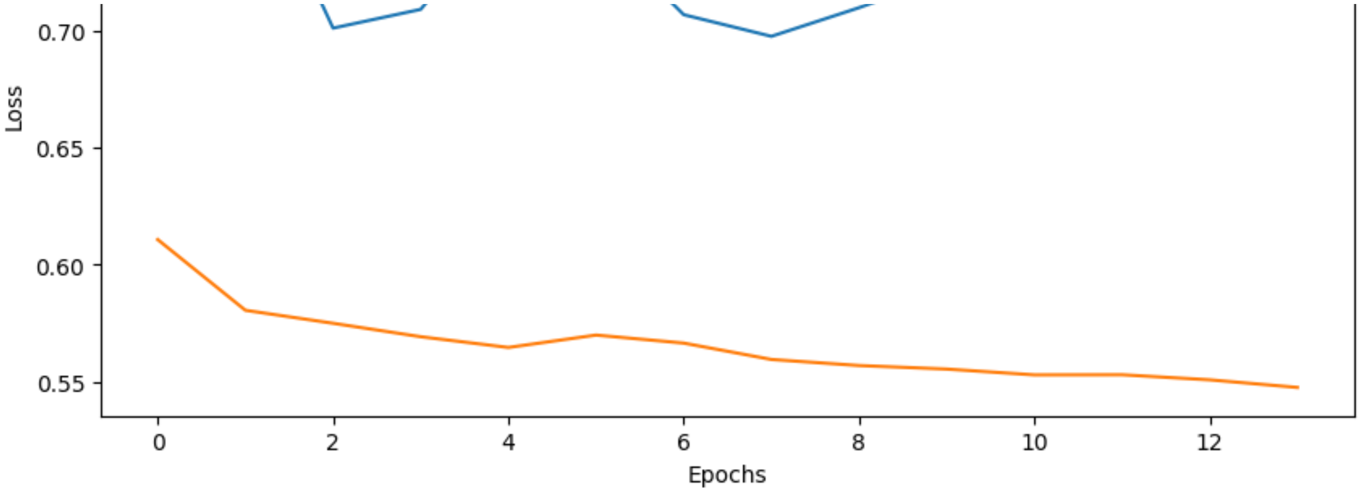
```
Epoch: 0, Train loss: 0.6111992721116976, Train accuracy: 0.6658415794372559
Val loss: 0.7188661226204464, Val accuracy: 0.42725173210161665
Epoch: 1, Train loss: 0.576042109314758, Train accuracy: 0.6861386299133301
Val loss: 0.7080849366528645, Val accuracy: 0.42609699769053117
Epoch: 2, Train loss: 0.5752367729794468, Train accuracy: 0.683993399143219
Val loss: 0.7307723739317485, Val accuracy: 0.42494226327944573
Epoch: 3, Train loss: 0.5679179099526735, Train accuracy: 0.6798679828643799
Val loss: 0.727758709873472, Val accuracy: 0.42609699769053117
Epoch: 4, Train loss: 0.5719957719541618, Train accuracy: 0.6797029972076416
Val loss: 0.7438280561140606, Val accuracy: 0.5265588914549654
Epoch: 5, Train loss: 0.5581241639533846, Train accuracy: 0.6854785680770874
Val loss: 0.7180214481694356, Val accuracy: 0.43187066974595845
Epoch: 6, Train loss: 0.5585771786104334, Train accuracy: 0.6863036155700684
Val loss: 0.7089483929531915, Val accuracy: 0.42609699769053117
Epoch: 7, Train loss: 0.5513414288904801, Train accuracy: 0.696864664554596
Val loss: 0.7359373228890554, Val accuracy: 0.42609699769053117
Early stopped training at epoch 7
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = art_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_art.pth"
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
```

```
Epoch: 0, Train loss: 0.6107877578565223, Train accuracy: 0.668722927570343
Val loss: 0.7577815806424177, Val accuracy: 0.4305717619603267
Epoch: 1, Train loss: 0.5806058784372292, Train accuracy: 0.6870623826980591
Val loss: 0.7817030129609284, Val accuracy: 0.4305717619603267
Epoch: 2, Train loss: 0.5750590524461994, Train accuracy: 0.6902300715446472
Val loss: 0.7010618669015388, Val accuracy: 0.43990665110851807
Epoch: 3, Train loss: 0.569317376983289, Train accuracy: 0.6832277178764343
Val loss: 0.7090758129402444, Val accuracy: 0.4305717619603267
Epoch: 4, Train loss: 0.5647323350582014, Train accuracy: 0.6918972730636597
Val loss: 0.7547492760199087, Val accuracy: 0.4305717619603267
Epoch: 5, Train loss: 0.5700235910199412, Train accuracy: 0.6867288947105408
Val loss: 0.7442822169374536, Val accuracy: 0.4305717619603267
Epoch: 6, Train loss: 0.5665985011048776, Train accuracy: 0.6832277178764343
Val loss: 0.7067962310932301, Val accuracy: 0.4305717619603267
Epoch: 7, Train loss: 0.5596084717155576, Train accuracy: 0.6905634999275208
Val loss: 0.6975613435109456, Val accuracy: 0.44457409568261375
Epoch: 8, Train loss: 0.5570168043025615, Train accuracy: 0.6952317357063293
Val loss: 0.7097838079487836, Val accuracy: 0.4364060676779463
Epoch: 9, Train loss: 0.5555071741113984, Train accuracy: 0.6943981051445007
Val loss: 0.7232455169713057, Val accuracy: 0.45507584597432904
Epoch: 10, Train loss: 0.5530311736455557, Train accuracy: 0.6978992819786072
Val loss: 0.7895304229524399, Val accuracy: 0.4457409568261377
Epoch: 11, Train loss: 0.5530786467576353, Train accuracy: 0.694231390953064
Val loss: 0.7123705921349704, Val accuracy: 0.4632438739789965
Epoch: 12, Train loss: 0.5509157304507011, Train accuracy: 0.698065996170044
Val loss: 0.715651344369959, Val accuracy: 0.45857642940490084
Epoch: 13, Train loss: 0.5476686119059397, Train accuracy: 0.7029009461402893
Val loss: 0.734109496287968, Val accuracy: 0.45857642940490084
Early stopped training at epoch 13
```



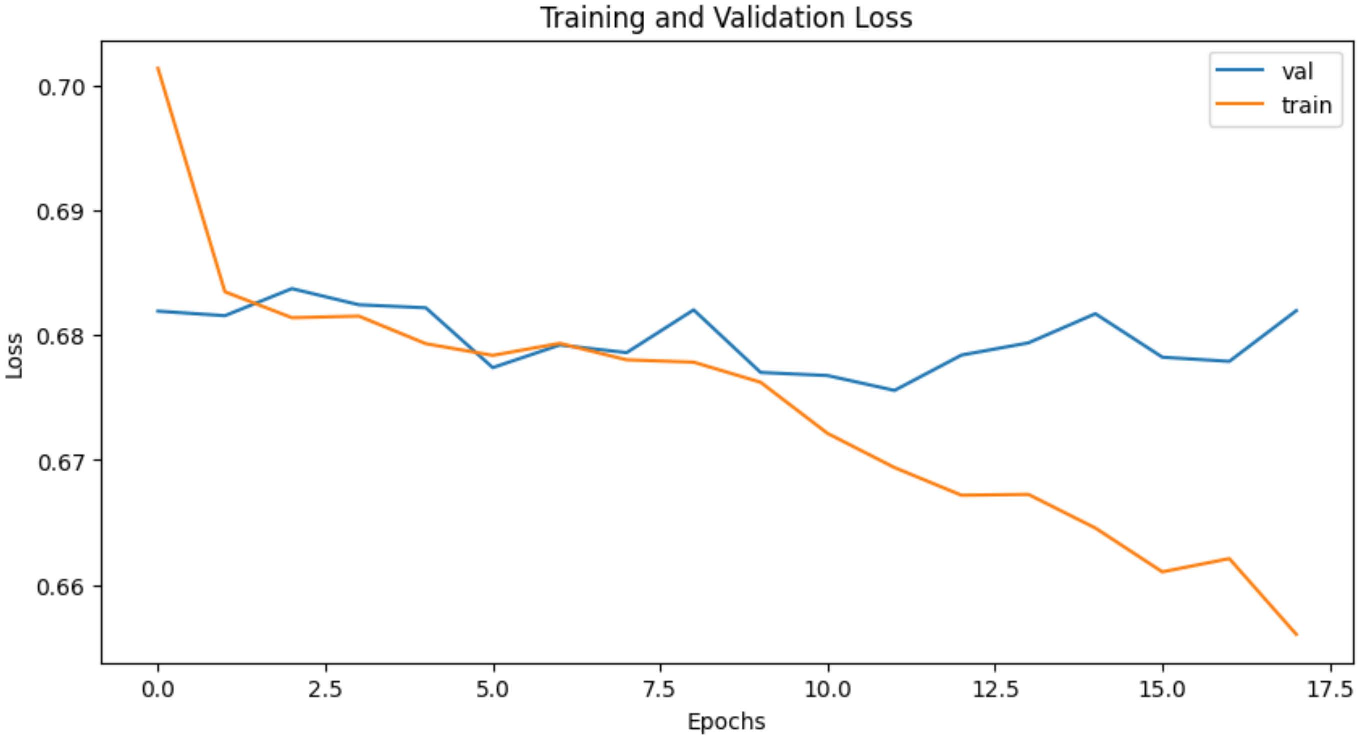


## Test model limited to EEG data

```
class eeg_model(nn.Module):
  # use this class to define your model
  def __init__(self):

    super().__init__()

    self.encoder_eeg = nn.Sequential(
                    nn.Conv1d(1,1,3,stride=1, padding='same')
                    )

    self.layer1_eeg = nn.Sequential(
                    nn.BatchNorm1d(1),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(1,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

    self.maxpool_eeg_layer1 = nn.MaxPool1d(2, 2)

    self.layer2_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

    self.layer3_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

    self.maxpool_eeg_layer3 = nn.MaxPool1d(2, 2)

    self.layer4_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

    self.layer5_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

    self.maxpool_eeg_layer5 = nn.MaxPool1d(2, 2)

    self.layer6_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

    self.layer7_eeg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,3,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Conv1d(4,4,3,stride=1, padding='same')
                    )

    self.maxpool_eeg_layer7 = nn.MaxPool1d(2, 2)

    self.layer8_eeg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,3,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Conv1d(4,4,3,stride=1, padding='same')
                    )

    self.layer9_eeg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,3,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                    nn.Conv1d(4,4,3,stride=1, padding='same')
                    )

    self.maxpool_eeg_layer9 = nn.MaxPool1d(2, 2)

    self.layer10_eeg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(4,6,3,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Conv1d(6,6,3,stride=1, padding='same')
                        )

    self.layer11_eeg = nn.Sequential(
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(6,6,3,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Conv1d(6,6,3,stride=1, padding='same')
                        )

    self.maxpool_eeg_layer11 = nn.MaxPool1d(2, 2)

    self.layer12_eeg = nn.Sequential(
                        nn.BatchNorm1d(6),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(6,6,3,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.ReLU()
                        )

    self.linear_eeg = nn.Linear(120*6, 96)

    self.linear_combined1 = nn.Linear(96, 16)
    self.linear_combined2 = nn.Linear(16, 1)

    self.apply(self._init_weights)

  def _init_weights(self, module):

      if isinstance(module, nn.Conv1d):
        nn.init.kaiming_normal_(module.weight, nonlinearity='relu')

        if module.bias is not None:
          nn.init.zeros_(module.bias)

      elif isinstance(module, nn.BatchNorm1d):
        nn.init.constant_(module.weight, 1)
        nn.init.constant_(module.bias, 0)

      elif isinstance(module, nn.Linear):
        nn.init.xavier_normal_(module.weight)

        if module.bias is not None:
          nn.init.zeros_(module.bias)

  def forward(self, ecg, art, eeg):

    eeg = self.encoder_eeg(eeg)
    tmp = self.layer1_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer1(eeg)
    tmp = self.layer2_eeg(eeg)
    eeg = tmp + eeg
    tmp = self.layer3_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer3(eeg)
    tmp = self.layer4_eeg(eeg)
    eeg = tmp + eeg
    tmp = self.layer5_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer5(eeg)
    eeg = self.layer6_eeg(eeg)
    tmp = self.layer7_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer7(eeg)
    tmp = self.layer8_eeg(eeg)
    eeg = tmp + eeg
    tmp = self.layer9_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer9(eeg)
    eeg = self.layer10_eeg(eeg)
    tmp = self.layer11_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer11(eeg)
    tmp = self.layer12_eeg(eeg)
    eeg = tmp + eeg
    eeg = torch.flatten(eeg, 1)
    eeg = F.relu(self.linear_eeg(eeg))

    combined = F.relu(self.linear_combined1(eeg))
    logits = self.linear_combined2(combined)
    # probs = F.sigmoid(logits)
    return logits


loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = eeg_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_eeg.pth"
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
```

```
Epoch: 0, Train loss: 0.7013491169478988, Train accuracy: 0.557284951210022
Val loss: 0.6819134844200953, Val accuracy: 0.5842696629213483
Epoch: 1, Train loss: 0.6834645690599669, Train accuracy: 0.5638639330863953
Val loss: 0.6815498215811595, Val accuracy: 0.5853932584269663
Epoch: 2, Train loss: 0.6813890324936599, Train accuracy: 0.5640243887901306
Val loss: 0.6837170911686761, Val accuracy: 0.5853932584269663
Epoch: 3, Train loss: 0.6815152188932788, Train accuracy: 0.5640243887901306
Val loss: 0.6824257799557278, Val accuracy: 0.5853932584269663
Epoch: 4, Train loss: 0.6793082985400541, Train accuracy: 0.5637034773826599
Val loss: 0.682181641459465, Val accuracy: 0.5853932584269663
Epoch: 5, Train loss: 0.6783730895742678, Train accuracy: 0.5637034773826599
Val loss: 0.677392589194434, Val accuracy: 0.5853932584269663
Epoch: 6, Train loss: 0.6793361162343594, Train accuracy: 0.5637034773826599
Val loss: 0.6792021500212807, Val accuracy: 0.5853932584269663
Epoch: 7, Train loss: 0.6780165084213287, Train accuracy: 0.5637034773826599
Val loss: 0.6785920326198851, Val accuracy: 0.5853932584269663
Epoch: 8, Train loss: 0.677829888038366, Train accuracy: 0.5637034773826599
Val loss: 0.6820160320826939, Val accuracy: 0.5853932584269663
Epoch: 9, Train loss: 0.6762216066671428, Train accuracy: 0.5633825659751892
Val loss: 0.6770081222057343, Val accuracy: 0.5853932584269663
Epoch: 10, Train loss: 0.6721289086709127, Train accuracy: 0.5637034773826599
Val loss: 0.6767573697226388, Val accuracy: 0.5853932584269663
Epoch: 11, Train loss: 0.6693924292061234, Train accuracy: 0.5640243887901306
Val loss: 0.675568572112492, Val accuracy: 0.5865168539325842
Epoch: 12, Train loss: 0.6671785938724474, Train accuracy: 0.5635430216789246
Val loss: 0.6783951159034459, Val accuracy: 0.5853932584269663
Epoch: 13, Train loss: 0.6672424039730518, Train accuracy: 0.5792682766914368
Val loss: 0.6793746926954815, Val accuracy: 0.5831460674157304
Epoch: 14, Train loss: 0.6645530215123805, Train accuracy: 0.5794287323951721
Val loss: 0.681707895227841, Val accuracy: 0.5865168539325842
Epoch: 15, Train loss: 0.6610431505871676, Train accuracy: 0.5805519819259644
Val loss: 0.6782277545758656, Val accuracy: 0.5910112359550562
Epoch: 16, Train loss: 0.6621045868112125, Train accuracy: 0.5815147757530212
```

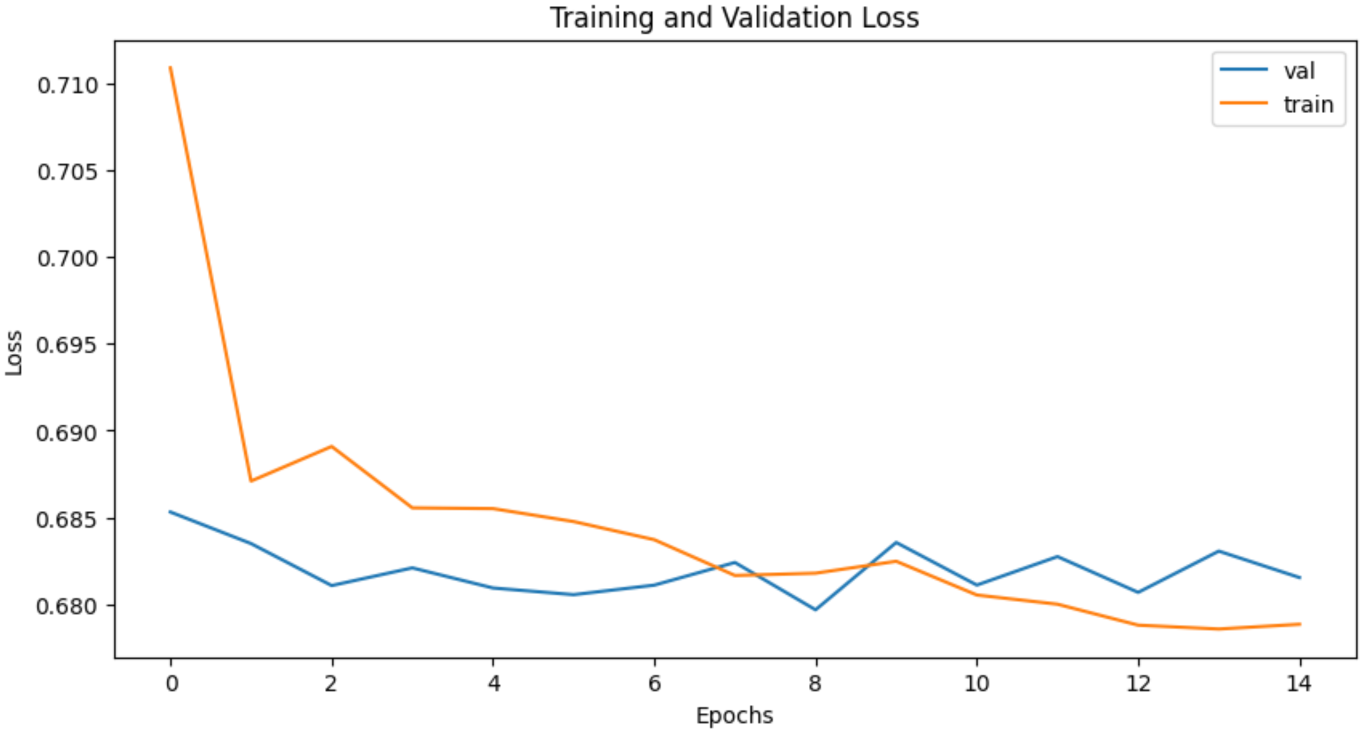
```
Val loss: 0.6778928509780339, Val accuracy: 0.5865168539325842
Epoch: 17, Train loss: 0.6560447387425982, Train accuracy: 0.591142475605011
Val loss: 0.68195122054645, Val accuracy: 0.5820224719101124
Early stopped training at epoch 17
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = eeg_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_eeg.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
```

```
Epoch: 0, Train loss: 0.7109000494235355, Train accuracy: 0.5573160648345947
Val loss: 0.6853079135928835, Val accuracy: 0.5859728506787331
Epoch: 1, Train loss: 0.687089999905496, Train accuracy: 0.5618431568145752
Val loss: 0.6834887393883295, Val accuracy: 0.5882352941176471
Epoch: 2, Train loss: 0.6890858260488548, Train accuracy: 0.5610347390174866
```

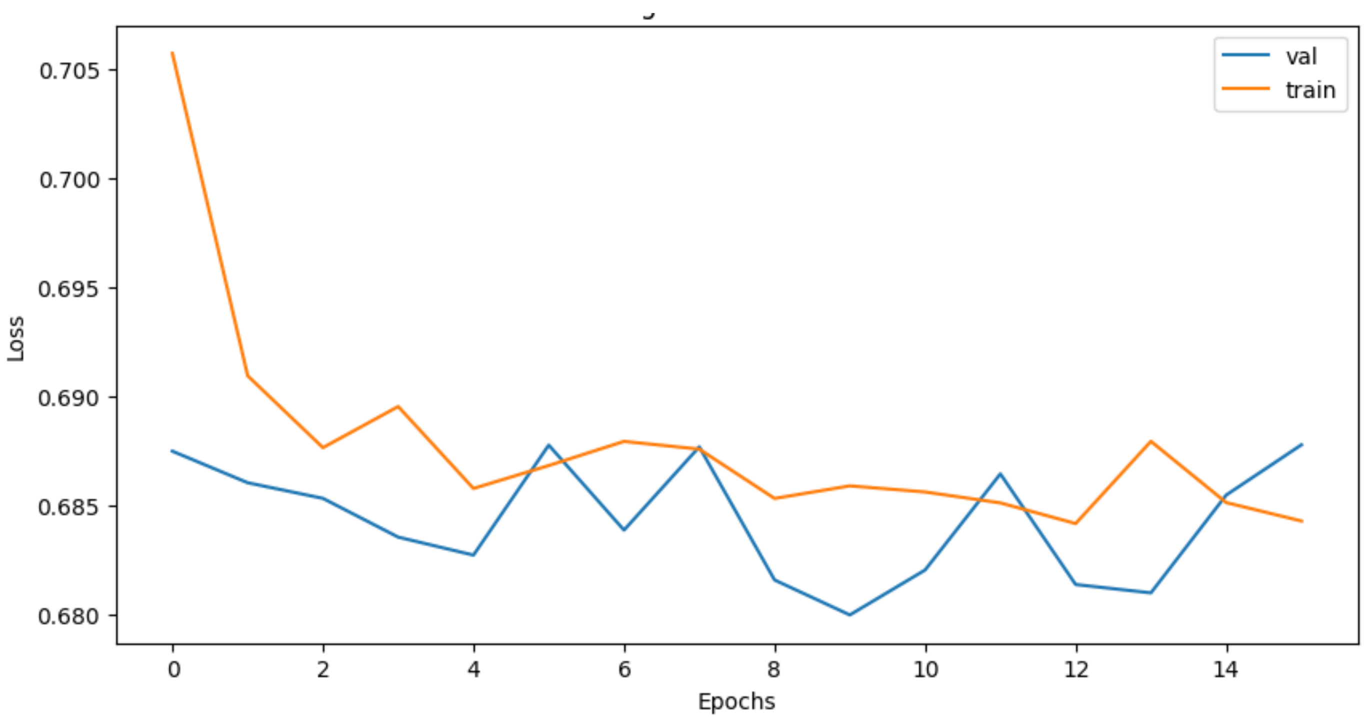
```
Val loss: 0.6810684757573263, Val accuracy: 0.5859728506787331
Epoch: 3, Train loss: 0.6855460804320191, Train accuracy: 0.5632982850074768
Val loss: 0.6820871127503259, Val accuracy: 0.5859728506787331
Epoch: 4, Train loss: 0.6855035671131594, Train accuracy: 0.5616815090179443
Val loss: 0.6809287241527011, Val accuracy: 0.5859728506787331
Epoch: 5, Train loss: 0.6847570991554661, Train accuracy: 0.5637833476066589
Val loss: 0.6805439612695151, Val accuracy: 0.5859728506787331
Epoch: 6, Train loss: 0.6837116937244141, Train accuracy: 0.563136637210846
Val loss: 0.6810929200478963, Val accuracy: 0.5859728506787331
Epoch: 7, Train loss: 0.6816506555510222, Train accuracy: 0.5636216402053833
Val loss: 0.6823994751487461, Val accuracy: 0.5871040723981901
Epoch: 8, Train loss: 0.6817815571131972, Train accuracy: 0.5658851861953735
Val loss: 0.6796713705573764, Val accuracy: 0.5871040723981901
Epoch: 9, Train loss: 0.6824682232248465, Train accuracy: 0.5645917654037476
Val loss: 0.6835520203624453, Val accuracy: 0.580316742081448
Epoch: 10, Train loss: 0.6805237535420404, Train accuracy: 0.5647534132003784
Val loss: 0.6810961174113409, Val accuracy: 0.582579185520362
Epoch: 11, Train loss: 0.6799938412761148, Train accuracy: 0.5654001832008362
Val loss: 0.6827384829521179, Val accuracy: 0.581447963800905
Epoch: 12, Train loss: 0.6787871293512775, Train accuracy: 0.5652384757995605
Val loss: 0.6806763453142983, Val accuracy: 0.5780542986425339
Epoch: 13, Train loss: 0.6785641132676996, Train accuracy: 0.5675020217895508
Val loss: 0.6830534892422814, Val accuracy: 0.582579185520362
Epoch: 14, Train loss: 0.6788371814578988, Train accuracy: 0.5684720873832703
Val loss: 0.6815312462193625, Val accuracy: 0.580316742081448
Early stopped training at epoch 14
```



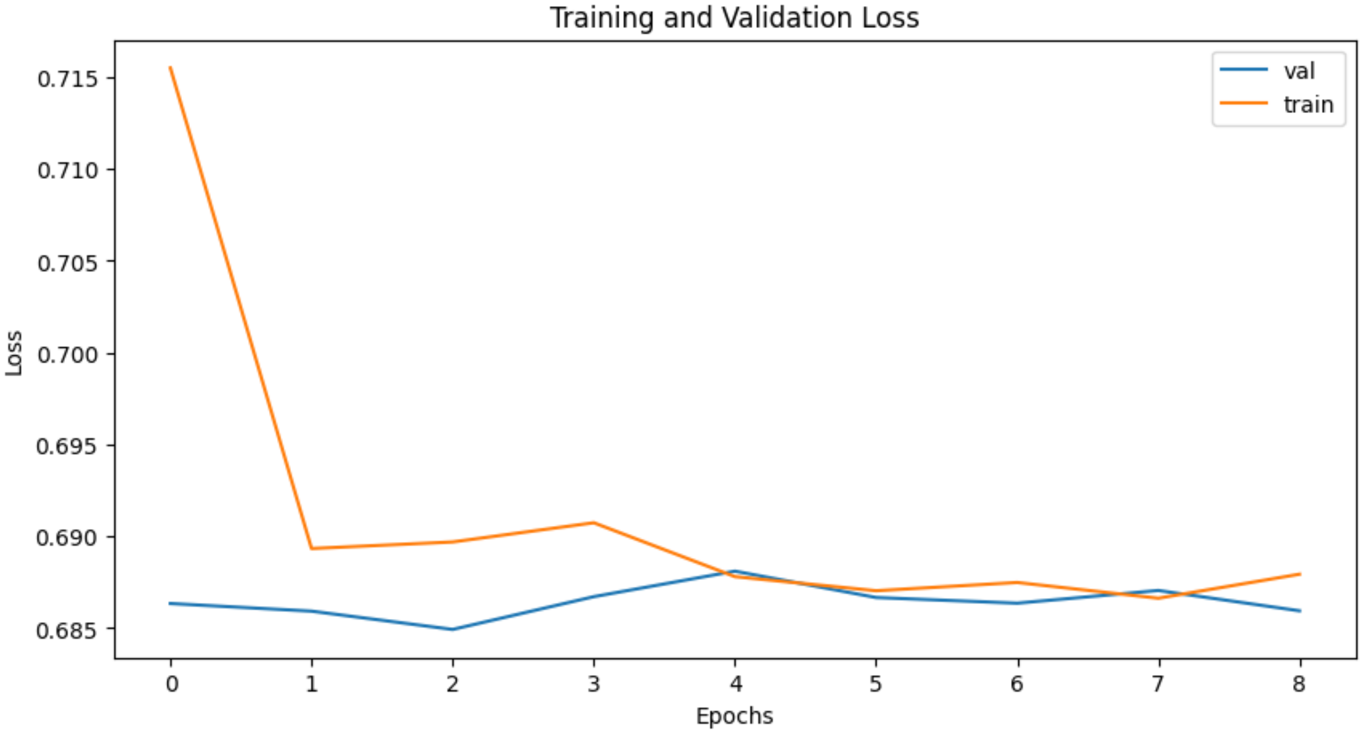
```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 10 min model
torch.manual_seed(seed)
model = eeg_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_eeg.pth"
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
```

```
Epoch: 0, Train loss: 0.7057406730384324, Train accuracy: 0.550990104675293
Val loss: 0.6874748830284393, Val accuracy: 0.5704387990762124
Epoch: 1, Train loss: 0.6909392985967127, Train accuracy: 0.551980197429657
Val loss: 0.6860256961413792, Val accuracy: 0.5739030023094688
Epoch: 2, Train loss: 0.6876367532774167, Train accuracy: 0.5523102283477783
Val loss: 0.685307753937585, Val accuracy: 0.5739030023094688
Epoch: 3, Train loss: 0.689525294697324, Train accuracy: 0.5537953972816467
Val loss: 0.6835276995386398, Val accuracy: 0.5739030023094688
Epoch: 4, Train loss: 0.6857615721107709, Train accuracy: 0.551980197429657
Val loss: 0.6827017728771482, Val accuracy: 0.5739030023094688
Epoch: 5, Train loss: 0.6868155106852943, Train accuracy: 0.5547854900360107
Val loss: 0.6877541818789075, Val accuracy: 0.5739030023094688
Epoch: 6, Train loss: 0.6879240959390949, Train accuracy: 0.5526402592658997
Val loss: 0.6838498988321849, Val accuracy: 0.5750577367205543
Epoch: 7, Train loss: 0.6875681196895763, Train accuracy: 0.5557755827903748
Val loss: 0.6876733707530158, Val accuracy: 0.5739030023094688
Epoch: 8, Train loss: 0.6853036172712597, Train accuracy: 0.551980197429657
Val loss: 0.6815596265452248, Val accuracy: 0.569284064665127
Epoch: 9, Train loss: 0.6858818007381049, Train accuracy: 0.5544554591178894
Val loss: 0.6799535176583699, Val accuracy: 0.5739030023094688
Epoch: 10, Train loss: 0.6856011503993875, Train accuracy: 0.5561056137084961
Val loss: 0.6820180330957685, Val accuracy: 0.5727482678983834
Epoch: 11, Train loss: 0.6850995318330947, Train accuracy: 0.5564356446266174
Val loss: 0.6864319401127953, Val accuracy: 0.5715935334872979
Epoch: 12, Train loss: 0.6841509525925413, Train accuracy: 0.5542904138565063
Val loss: 0.6813512359346662, Val accuracy: 0.5727482678983834
Epoch: 13, Train loss: 0.687926984895574, Train accuracy: 0.5592409372329712
Val loss: 0.6809679376227515, Val accuracy: 0.5739030023094688
Epoch: 14, Train loss: 0.6851143374694849, Train accuracy: 0.5570957064628601
Val loss: 0.6854594669171743, Val accuracy: 0.5715935334872979
Epoch: 15, Train loss: 0.6842675916825978, Train accuracy: 0.5577557682991028
Val loss: 0.6877682060003281, Val accuracy: 0.5750577367205543
Early stopped training at epoch 15
```

Training and Validation Loss



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = eeg_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_eeg.pth"
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
Epoch: 0, Train loss: 0.7154928285426082, Train accuracy: 0.5365121960639954
Val loss: 0.6863476898935108, Val accuracy: 0.5694282380396732
Epoch: 1, Train loss: 0.6893400026782825, Train accuracy: 0.5483494400978088
Val loss: 0.6859320446296974, Val accuracy: 0.5694282380396732
Epoch: 2, Train loss: 0.6896974200723807, Train accuracy: 0.5498499274253845
Val loss: 0.6849361393186781, Val accuracy: 0.574095682613769
Epoch: 3, Train loss: 0.6907424383578439, Train accuracy: 0.5485161542892456
Val loss: 0.6867197972756843, Val accuracy: 0.5705950991831972
Epoch: 4, Train loss: 0.6878101604109647, Train accuracy: 0.5493497848510742
Val loss: 0.6881077444111858, Val accuracy: 0.5694282380396732
Epoch: 5, Train loss: 0.6870432000392356, Train accuracy: 0.5501834154129028
Val loss: 0.6866740032478614, Val accuracy: 0.5694282380396732
Epoch: 6, Train loss: 0.6874922303884734, Train accuracy: 0.5501834154129028
Val loss: 0.6863658472343729, Val accuracy: 0.5694282380396732
Epoch: 7, Train loss: 0.6866353576920278, Train accuracy: 0.5516839027404785
Val loss: 0.6870568924480014, Val accuracy: 0.5670945157526255
Epoch: 8, Train loss: 0.6879422621513931, Train accuracy: 0.5503501296043396
Val loss: 0.6859513388739689, Val accuracy: 0.5682613768961493
Early stopped training at epoch 8
```



## Test model with only 6 residual blocks instead of 12

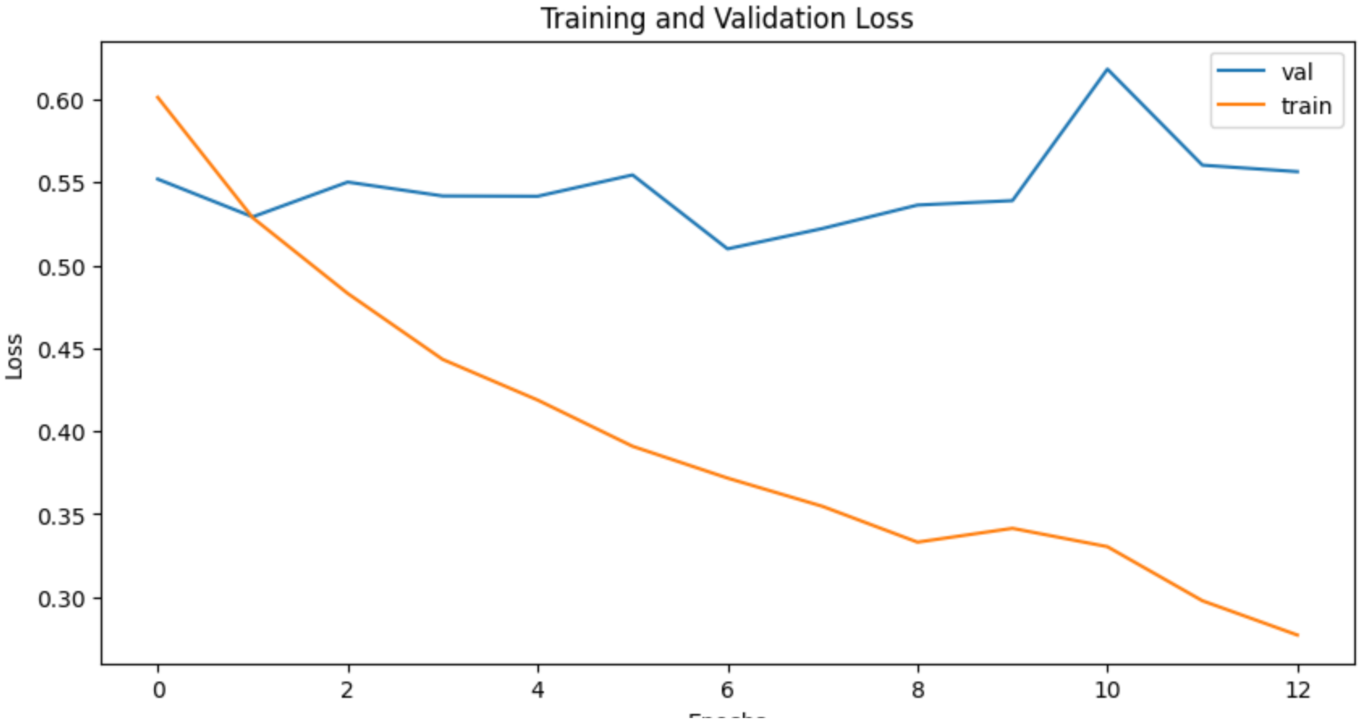
```
class shallow_model(nn.Module):
  # use this class to define your model
  def __init__(self):

    super().__init__()

    self.encoder_ecg = nn.Sequential(
                        nn.Conv1d(in_channels=1, out_channels=1, kernel_size=3,st
                        )

    self.layer1_ecg = nn.Sequential(
                        nn.BatchNorm1d(1),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(1,2,15,stride=1, padding='same'),
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Conv1d(2,2,15,stride=1, padding='same')
                        )

    self.maxpool_ecg_layer1 = nn.MaxPool1d(2, 2)

    self.layer2_ecg = nn.Sequential(
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(2,2,15,stride=1, padding='same'),
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Conv1d(2,2,15,stride=1, padding='same')
                        )

    self.layer3_ecg = nn.Sequential(
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Dropout(),
                        nn.Conv1d(2,2,15,stride=1, padding='same'),
                        nn.BatchNorm1d(2),
                        nn.ReLU(),
                        nn.Conv1d(2,2,15,stride=1, padding='same')
                        )

    self.maxpool_ecg_layer3 = nn.MaxPool1d(2, 2)
```

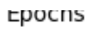
```
        self.layer4_ecg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.layer5_ecg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_ecg_layer5 = nn.MaxPool1d(2, 2)

        self.layer6_ecg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,4,15,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU()
                            )

        self.linear_ecg = nn.Linear(3750*4, 32)

        self.encoder_art = nn.Sequential(
                            nn.Conv1d(1,1,3,stride=1, padding='same')
                            )

        self.layer1_art = nn.Sequential(
                            nn.BatchNorm1d(1),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(1,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_art_layer1 = nn.MaxPool1d(2, 2)

        self.layer2_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.layer3_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.maxpool_art_layer3 = nn.MaxPool1d(2, 2)

        self.layer4_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )

        self.layer5_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,15,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,15,stride=1, padding='same')
                            )
        self.maxpool_art_layer5 = nn.MaxPool1d(2, 2)

        self.layer6_art = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,4,15,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU()
                            )

        self.linear_art = nn.Linear(3750*4, 32)

        self.encoder_eeg = nn.Sequential(
                            nn.Conv1d(1,1,3,stride=1, padding='same')
                            )

        self.layer1_eeg = nn.Sequential(
                            nn.BatchNorm1d(1),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(1,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer1 = nn.MaxPool1d(2, 2)

        self.layer2_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.layer3_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer3 = nn.MaxPool1d(2, 2)

        self.layer4_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.layer5_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,2,7,stride=1, padding='same'),
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Conv1d(2,2,7,stride=1, padding='same')
                            )

        self.maxpool_eeg_layer5 = nn.MaxPool1d(2, 2)

        self.layer6_eeg = nn.Sequential(
                            nn.BatchNorm1d(2),
                            nn.ReLU(),
                            nn.Dropout(),
                            nn.Conv1d(2,4,7,stride=1, padding='same'),
                            nn.BatchNorm1d(4),
                            nn.ReLU()
                            )

        self.linear_eeg = nn.Linear(960*4, 32)

        self.linear_combined1 = nn.Linear(96, 16)
        self.linear_combined2 = nn.Linear(16, 1)

        self.apply(self._init_weights)
```

```
  def _init_weights(self, module):

      if isinstance(module, nn.Conv1d):
        nn.init.kaiming_normal_(module.weight, nonlinearity='relu')

        if module.bias is not None:
          nn.init.zeros_(module.bias)

      elif isinstance(module, nn.BatchNorm1d):
        nn.init.constant_(module.weight, 1)
        nn.init.constant_(module.bias, 0)

      elif isinstance(module, nn.Linear):
        nn.init.xavier_normal_(module.weight)

        if module.bias is not None:
          nn.init.zeros_(module.bias)

  def forward(self, ecg, art, eeg):

    ecg = self.encoder_ecg(ecg)
    tmp = self.layer1_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer1(ecg)
    tmp = self.layer2_ecg(ecg)
    ecg = tmp + ecg
    tmp = self.layer3_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer3(ecg)
    tmp = self.layer4_ecg(ecg)
    ecg = tmp + ecg
    tmp = self.layer5_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer5(ecg)
    ecg = self.layer6_ecg(ecg)
    ecg = torch.flatten(ecg, 1)
    ecg = F.relu(self.linear_ecg(ecg))

    art = self.encoder_art(art)
    tmp = self.layer1_art(art)
    art = tmp + art
    art = self.maxpool_art_layer1(art)
    tmp = self.layer2_art(art)
    art = tmp + art
    tmp = self.layer3_art(art)
    art = tmp + art
    art = self.maxpool_art_layer3(art)
    tmp = self.layer4_art(art)
    art = tmp + art
    tmp = self.layer5_art(art)
    art = tmp + art
    art = self.maxpool_art_layer5(art)
    art = self.layer6_art(art)
    art = torch.flatten(art, 1)
    art = F.relu(self.linear_art(art))

    eeg = self.encoder_eeg(eeg)
    tmp = self.layer1_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer1(eeg)
    tmp = self.layer2_eeg(eeg)
    eeg = tmp + eeg
    tmp = self.layer3_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer3(eeg)
    tmp = self.layer4_eeg(eeg)
    eeg = tmp + eeg
    tmp = self.layer5_eeg(eeg)
    eeg = tmp + eeg
    eeg = self.maxpool_eeg_layer5(eeg)
    eeg = self.layer6_eeg(eeg)
    eeg = torch.flatten(eeg, 1)
    eeg = F.relu(self.linear_eeg(eeg))

    combined = F.relu(self.linear_combined1(torch.cat((ecg, art, eeg), -1)))
    logits = self.linear_combined2(combined)
    # probs = F.sigmoid(logits)

    return logits


loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = shallow_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
```

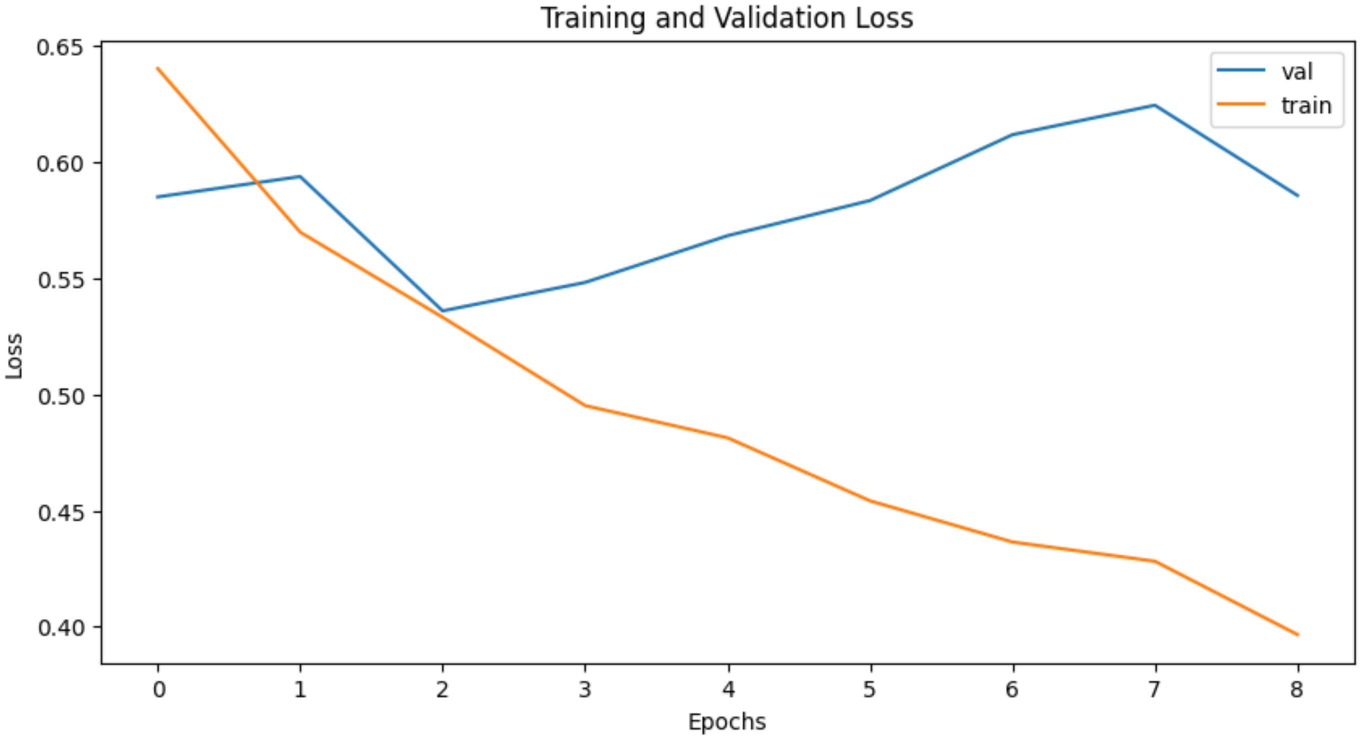
```
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_shallow.pth"
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
```

```
Epoch: 0, Train loss: 0.6012184792825898, Train accuracy: 0.7155006527900696
Val loss: 0.5519191271492415, Val accuracy: 0.7179775280898877
Epoch: 1, Train loss: 0.5286384480449446, Train accuracy: 0.744544267654419
Val loss: 0.5291716403194836, Val accuracy: 0.7224719101123596
Epoch: 2, Train loss: 0.483014011566691, Train accuracy: 0.7673299312591553
Val loss: 0.5501368695071766, Val accuracy: 0.7123595505617978
Epoch: 3, Train loss: 0.4432426324169198, Train accuracy: 0.7920410633087158
Val loss: 0.5417706593871116, Val accuracy: 0.7191011235955056
Epoch: 4, Train loss: 0.418740955335644, Train accuracy: 0.807284951210022
Val loss: 0.5415825982178961, Val accuracy: 0.7303370786516854
Epoch: 5, Train loss: 0.39086533407650803, Train accuracy: 0.8255776762962341
Val loss: 0.554443020905767, Val accuracy: 0.7168539325842697
Epoch: 6, Train loss: 0.371687150430006, Train accuracy: 0.8348844647407532
Val loss: 0.5098751272474018, Val accuracy: 0.7528089887640449
Epoch: 7, Train loss: 0.35464747546573044, Train accuracy: 0.8474004864692688
Val loss: 0.5221568739839962, Val accuracy: 0.7561797752808989
Epoch: 8, Train loss: 0.33307333242755494, Train accuracy: 0.8544608354568481
Val loss: 0.5362833355154311, Val accuracy: 0.7460674157303371
Epoch: 9, Train loss: 0.34139952859309275, Train accuracy: 0.8602374792098999
Val loss: 0.5389583387545177, Val accuracy: 0.7573033707865169
Epoch: 10, Train loss: 0.33033890301058016, Train accuracy: 0.860077023506164
Val loss: 0.6182500230414525, Val accuracy: 0.701123595505618
Epoch: 11, Train loss: 0.297738988722396, Train accuracy: 0.881899893283844
Val loss: 0.5603388526609966, Val accuracy: 0.751685393258427
Epoch: 12, Train loss: 0.27707117099664025, Train accuracy: 0.889120638370513
Val loss: 0.5564181464059013, Val accuracy: 0.7561797752808989
Early stopped training at epoch 12
```



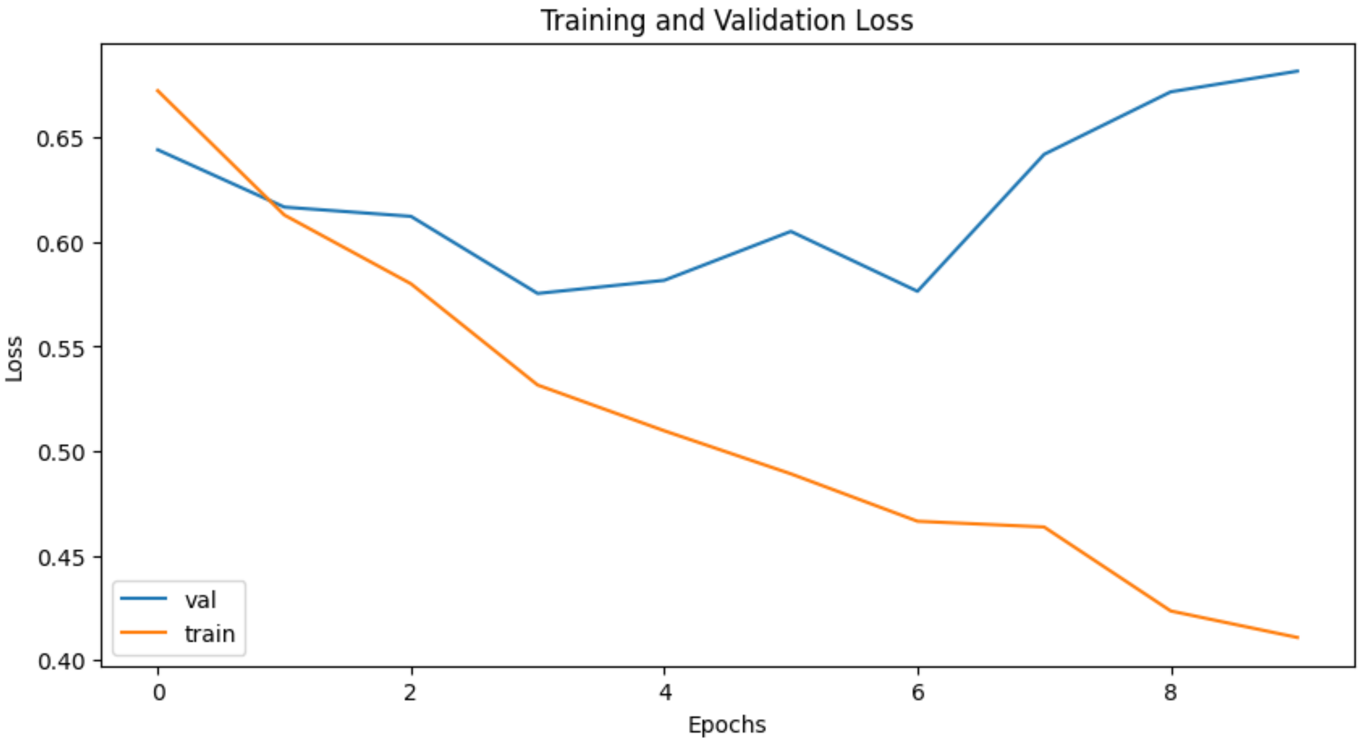


```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = shallow_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_shallow.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.6401503917760394, Train accuracy: 0.660953938961029
Val loss: 0.5849401652812957, Val accuracy: 0.667420814479638
Epoch: 1, Train loss: 0.5697552839979204, Train accuracy: 0.7065480947494507
Val loss: 0.593686596623489, Val accuracy: 0.6628959276018099
Epoch: 2, Train loss: 0.5331256452450949, Train accuracy: 0.7337105870246887
Val loss: 0.5358969005090849, Val accuracy: 0.7217194570135747
Epoch: 3, Train loss: 0.49515299779711563, Train accuracy: 0.7590945959091187
Val loss: 0.5481418541499548, Val accuracy: 0.7183257918552036
Epoch: 4, Train loss: 0.4812868524966167, Train accuracy: 0.7641066908836365
Val loss: 0.5682425552180835, Val accuracy: 0.7386877828054299
Epoch: 5, Train loss: 0.4541748348897565, Train accuracy: 0.7783346772193909
Val loss: 0.5833732464483805, Val accuracy: 0.6877828054298643
Epoch: 6, Train loss: 0.4364600318761824, Train accuracy: 0.7912691831588745
Val loss: 0.6116932258009911, Val accuracy: 0.6990950226244343
Epoch: 7, Train loss: 0.42817242009657086, Train accuracy: 0.7980598211288452
Val loss: 0.6243852568524225, Val accuracy: 0.6923076923076923
Epoch: 8, Train loss: 0.3966366647613945, Train accuracy: 0.8200485110282898
Val loss: 0.5855712049773762, Val accuracy: 0.7194570135746606
Early stopped training at epoch 8
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
```

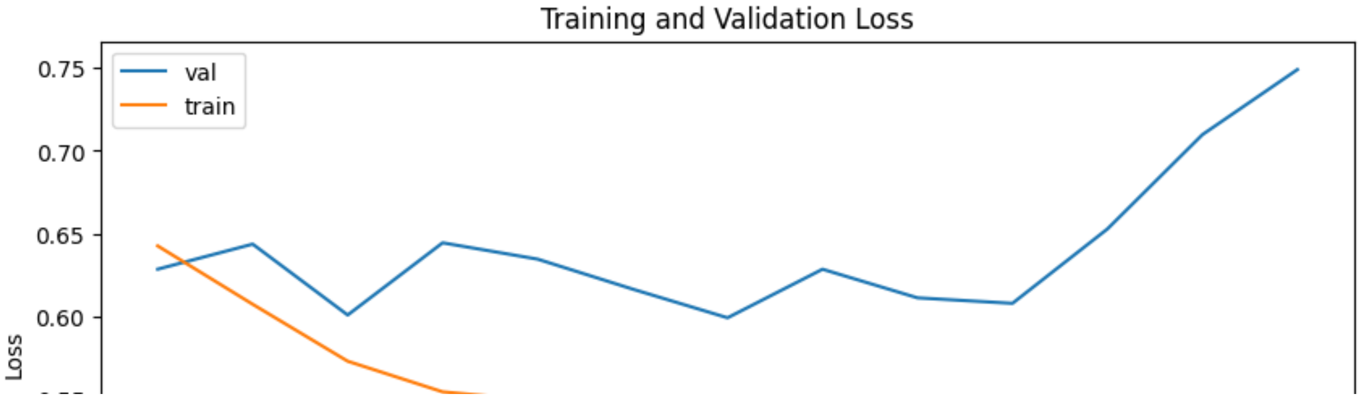
```
num_epoch = 100
early_stop_thresh = 5

# 10 min model
torch.manual_seed(seed)
model = shallow_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_shallow.pth"
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
Epoch: 0, Train loss: 0.6724914489406171, Train accuracy: 0.6196369528770447
Val loss: 0.6442126929759978, Val accuracy: 0.6362586605080831
Epoch: 1, Train loss: 0.6129806658615767, Train accuracy: 0.6603960394859314
Val loss: 0.6167825369962623, Val accuracy: 0.6466512702078522
Epoch: 2, Train loss: 0.5800665438962062, Train accuracy: 0.6848185062408447
Val loss: 0.6123498156666756, Val accuracy: 0.6812933025404158
Epoch: 3, Train loss: 0.5316092467544102, Train accuracy: 0.7212871313095093
Val loss: 0.5754363951938494, Val accuracy: 0.6651270207852193
Epoch: 4, Train loss: 0.5097023044875746, Train accuracy: 0.7419142127037048
Val loss: 0.5817254558205606, Val accuracy: 0.6501154734411085
Epoch: 5, Train loss: 0.4890830366131496, Train accuracy: 0.751980185508728
Val loss: 0.60515182358878, Val accuracy: 0.6374133949191686
Epoch: 6, Train loss: 0.46639017629938156, Train accuracy: 0.7699670195579529
Val loss: 0.5764772764274052, Val accuracy: 0.6859122401847575
Epoch: 7, Train loss: 0.46363954457512785, Train accuracy: 0.7699670195579529
Val loss: 0.6420444292681556, Val accuracy: 0.6351039260969977
Epoch: 8, Train loss: 0.4234309095557373, Train accuracy: 0.7960395812988281
Val loss: 0.6719596683979034, Val accuracy: 0.6270207852193995
Epoch: 9, Train loss: 0.4107466246822093, Train accuracy: 0.8057755827903748
Val loss: 0.6818661966494151, Val accuracy: 0.6327944572748267
Early stopped training at epoch 9
```

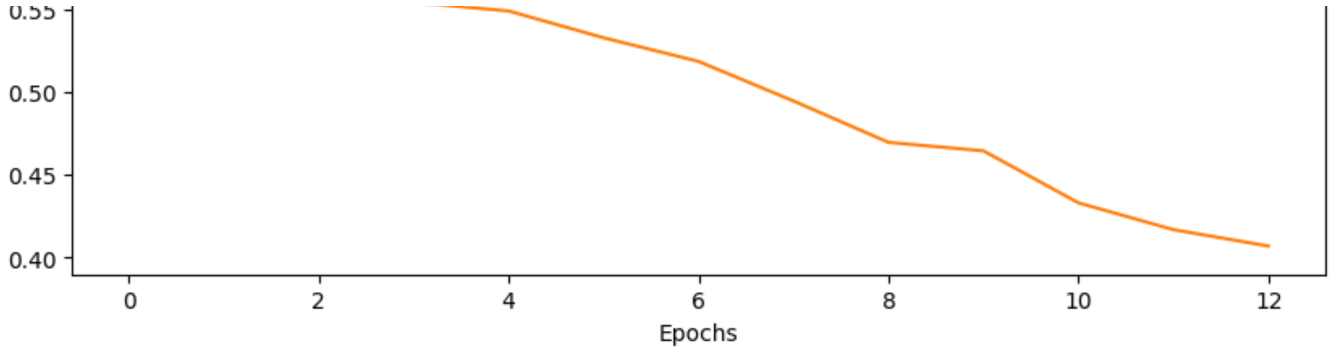


```
loss_func = nn.BCEWithLogitsLoss()
```

```
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = shallow_model()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_shallow.pth"
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
```

```
Epoch: 0, Train loss: 0.6424768049544436, Train accuracy: 0.6372123956680298
Val loss: 0.6284293764167362, Val accuracy: 0.6592765460910152
Epoch: 1, Train loss: 0.6072161260069351, Train accuracy: 0.6597198843955994
Val loss: 0.6435791607256288, Val accuracy: 0.6569428238039673
Epoch: 2, Train loss: 0.5728035232113695, Train accuracy: 0.6847282648086548
Val loss: 0.6007190898612694, Val accuracy: 0.6791131855309218
Epoch: 3, Train loss: 0.5542765300088662, Train accuracy: 0.700566828250885
Val loss: 0.6443653437826369, Val accuracy: 0.603267211201867
Epoch: 4, Train loss: 0.5490797105412676, Train accuracy: 0.706235408782959
Val loss: 0.6344943830260524, Val accuracy: 0.6849474912485414
Epoch: 5, Train loss: 0.5328776955743677, Train accuracy: 0.7222407460212708
Val loss: 0.6163847898995435, Val accuracy: 0.6861143523920653
Epoch: 6, Train loss: 0.5185894395816958, Train accuracy: 0.7275758385658264
Val loss: 0.5990440690958941, Val accuracy: 0.6732788798133023
Epoch: 7, Train loss: 0.4946766578582733, Train accuracy: 0.7507502436637878
Val loss: 0.6283974779976739, Val accuracy: 0.6826137689614936
Epoch: 8, Train loss: 0.46967122238967846, Train accuracy: 0.7624208331108093
Val loss: 0.6111053117999324, Val accuracy: 0.6592765460910152
Epoch: 9, Train loss: 0.4645385223056683, Train accuracy: 0.7737579345703125
Val loss: 0.6077814356044485, Val accuracy: 0.662777129521587
Epoch: 10, Train loss: 0.43309416647512305, Train accuracy: 0.790096700191497
Val loss: 0.6527439951896667, Val accuracy: 0.6674445740956826
Epoch: 11, Train loss: 0.41691319013921846, Train accuracy: 0.801433801651001
Val loss: 0.7097371739369853, Val accuracy: 0.6592765460910152
Epoch: 12, Train loss: 0.4070069853485962, Train accuracy: 0.8121040463447571
Val loss: 0.7490530764615093, Val accuracy: 0.6511085180863477
Early stopped training at epoch 12
```





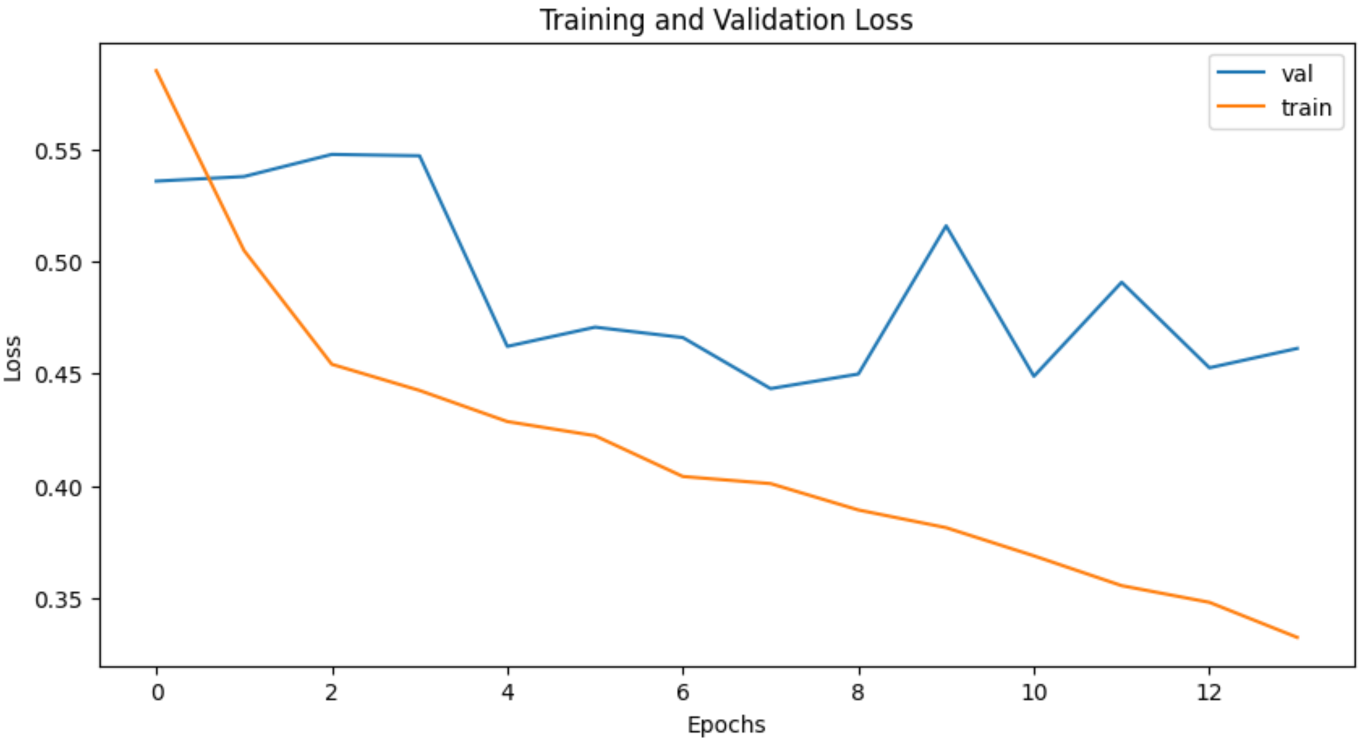
## Testing Leaky RELU instead of RELU

```
class model_leaky_relu(nn.Module):
  # use this class to define your model
  def __init__(self):

    super().__init__()

    self.encoder_ecg = nn.Sequential(
                    nn.Conv1d(in_channels=1, out_channels=1, kernel_size=3,st
                    )

    self.layer1_ecg = nn.Sequential(
                    nn.BatchNorm1d(1),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(1,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.maxpool_ecg_layer1 = nn.MaxPool1d(2, 2)

    self.layer2_ecg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.layer3_ecg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.maxpool_ecg_layer3 = nn.MaxPool1d(2, 2)

    self.layer4_ecg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.layer5_ecg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

    self.maxpool_ecg_layer5 = nn.MaxPool1d(2, 2)

    self.layer6_ecg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,4,15,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,15,stride=1, padding='same')
                    )

self.layer7_ecg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

self.maxpool_ecg_layer7 = nn.MaxPool1d(2, 2)

self.layer8_ecg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

self.layer9_ecg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

self.maxpool_ecg_layer9 = nn.MaxPool1d(2, 2)

self.layer10_ecg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,6,7,stride=1, padding='same'),
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    nn.Conv1d(6,6,7,stride=1, padding='same')
                    )

self.layer11_ecg = nn.Sequential(
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(6,6,7,stride=1, padding='same'),
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    nn.Conv1d(6,6,7,stride=1, padding='same')
                    )

self.maxpool_ecg_layer11 = nn.MaxPool1d(2, 2)

self.layer12_ecg = nn.Sequential(
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(6,6,7,stride=1, padding='same'),
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    )

self.linear_ecg = nn.Linear(468*6, 32)

self.encoder_art = nn.Sequential(
                    nn.Conv1d(1,1,3,stride=1, padding='same')
                    )

self.layer1_art = nn.Sequential(
                    nn.BatchNorm1d(1),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(1,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

self.maxpool_art_layer1 = nn.MaxPool1d(2, 2)

self.layer2_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

self.layer3_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

self.maxpool_art_layer3 = nn.MaxPool1d(2, 2)

self.layer4_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

self.layer5_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,15,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,15,stride=1, padding='same')
                    )

self.maxpool_art_layer5 = nn.MaxPool1d(2, 2)

self.layer6_art = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,4,15,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,15,stride=1, padding='same')
                    )

self.layer7_art = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

self.maxpool_art_layer7 = nn.MaxPool1d(2, 2)

self.layer8_art = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

self.layer9_art = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

self.maxpool_art_layer9 = nn.MaxPool1d(2, 2)

self.layer10_art = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,6,7,stride=1, padding='same'),
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    nn.Conv1d(6,6,7,stride=1, padding='same')
                    )

self.layer11_art = nn.Sequential(
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(6,6,7,stride=1, padding='same'),
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    nn.Conv1d(6,6,7,stride=1, padding='same')
                    )

self.maxpool_art_layer11 = nn.MaxPool1d(2, 2)

self.layer12_art = nn.Sequential(
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(6,6,7,stride=1, padding='same'),
                    nn.BatchNorm1d(6),
                    nn.LeakyReLU(),
                    )

self.linear_art = nn.Linear(468*6, 32)

self.encoder_eeg = nn.Sequential(
                    nn.Conv1d(1,1,3,stride=1, padding='same')
                    )

self.layer1_eeg = nn.Sequential(
                    nn.BatchNorm1d(1),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(1,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

self.maxpool_eeg_layer1 = nn.MaxPool1d(2, 2)
self.layer2_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

self.layer3_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

self.maxpool_eeg_layer3 = nn.MaxPool1d(2, 2)

self.layer4_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

self.layer5_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,2,7,stride=1, padding='same'),
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Conv1d(2,2,7,stride=1, padding='same')
                    )

self.maxpool_eeg_layer5 = nn.MaxPool1d(2, 2)

self.layer6_eeg = nn.Sequential(
                    nn.BatchNorm1d(2),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(2,4,7,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,7,stride=1, padding='same')
                    )

self.layer7_eeg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,3,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,3,stride=1, padding='same')
                    )

self.maxpool_eeg_layer7 = nn.MaxPool1d(2, 2)

self.layer8_eeg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,3,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,3,stride=1, padding='same')
                    )

self.layer9_eeg = nn.Sequential(
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Dropout(),
                    nn.Conv1d(4,4,3,stride=1, padding='same'),
                    nn.BatchNorm1d(4),
                    nn.LeakyReLU(),
                    nn.Conv1d(4,4,3,stride=1, padding='same')
                    )

self.maxpool_eeg_layer9 = nn.MaxPool1d(2, 2)

self.layer10_eeg = nn.Sequential(
                    nn.BatchNorm1d(4),
                        nn.LeakyReLU(),
                        nn.Dropout(),
                        nn.Conv1d(4,6,3,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.LeakyReLU(),
                        nn.Conv1d(6,6,3,stride=1, padding='same')
                        )

    self.layer11_eeg = nn.Sequential(
                        nn.BatchNorm1d(6),
                        nn.LeakyReLU(),
                        nn.Dropout(),
                        nn.Conv1d(6,6,3,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.LeakyReLU(),
                        nn.Conv1d(6,6,3,stride=1, padding='same')
                        )

    self.maxpool_eeg_layer11 = nn.MaxPool1d(2, 2)

    self.layer12_eeg = nn.Sequential(
                        nn.BatchNorm1d(6),
                        nn.LeakyReLU(),
                        nn.Dropout(),
                        nn.Conv1d(6,6,3,stride=1, padding='same'),
                        nn.BatchNorm1d(6),
                        nn.LeakyReLU(),
                        )

    self.linear_eeg = nn.Linear(120*6, 32)

    self.linear_combined1 = nn.Linear(96, 16)
    self.linear_combined2 = nn.Linear(16, 1)

    self.apply(self._init_weights)

  def _init_weights(self, module):

      if isinstance(module, nn.Conv1d):
        nn.init.kaiming_normal_(module.weight, nonlinearity='leaky_relu')

        if module.bias is not None:
          nn.init.zeros_(module.bias)

      elif isinstance(module, nn.BatchNorm1d):
        nn.init.constant_(module.weight, 1)
        nn.init.constant_(module.bias, 0)

      elif isinstance(module, nn.Linear):
        nn.init.xavier_normal_(module.weight)

        if module.bias is not None:
          nn.init.zeros_(module.bias)

  def forward(self, ecg, art, eeg):

    ecg = self.encoder_ecg(ecg)
    tmp = self.layer1_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer1(ecg)
    tmp = self.layer2_ecg(ecg)
    ecg = tmp + ecg
    tmp = self.layer3_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer3(ecg)
    tmp = self.layer4_ecg(ecg)
    ecg = tmp + ecg
    tmp = self.layer5_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer5(ecg)
    ecg = self.layer6_ecg(ecg)
    tmp = self.layer7_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer7(ecg)
    tmp = self.layer8_ecg(ecg)
    ecg = tmp + ecg
    tmp = self.layer9_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer9(ecg)
    ecg = self.layer10_ecg(ecg)
    tmp = self.layer11_ecg(ecg)
    ecg = tmp + ecg
    ecg = self.maxpool_ecg_layer11(ecg)
    tmp = self.layer12_ecg(ecg)
    ecg = tmp + ecg
    ecg = torch.flatten(ecg, 1)
    ecg = F.leaky_relu(self.linear_ecg(ecg))

    art = self.encoder_art(art)
    tmp = self.layer1_art(art)
```

```
        art = tmp + art
        art = self.maxpool_art_layer1(art)
        tmp = self.layer2_art(art)
        art = tmp + art
        tmp = self.layer3_art(art)
        art = tmp + art
        art = self.maxpool_art_layer3(art)
        tmp = self.layer4_art(art)
        art = tmp + art
        tmp = self.layer5_art(art)
        art = tmp + art
        art = self.maxpool_art_layer5(art)
        art = self.layer6_art(art)
        tmp = self.layer7_art(art)
        art = tmp + art
        art = self.maxpool_art_layer7(art)
        tmp = self.layer8_art(art)
        art = tmp + art
        tmp = self.layer9_art(art)
        art = tmp + art
        art = self.maxpool_art_layer9(art)
        art = self.layer10_art(art)
        tmp = self.layer11_art(art)
        art = tmp + art
        art = self.maxpool_art_layer11(art)
        tmp = self.layer12_art(art)
        art = tmp + art
        art = torch.flatten(art, 1)
        art = F.leaky_relu(self.linear_art(art))

        eeg = self.encoder_eeg(eeg)
        tmp = self.layer1_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer1(eeg)
        tmp = self.layer2_eeg(eeg)
        eeg = tmp + eeg
        tmp = self.layer3_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer3(eeg)
        tmp = self.layer4_eeg(eeg)
        eeg = tmp + eeg
        tmp = self.layer5_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer5(eeg)
        eeg = self.layer6_eeg(eeg)
        tmp = self.layer7_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer7(eeg)
        tmp = self.layer8_eeg(eeg)
        eeg = tmp + eeg
        tmp = self.layer9_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer9(eeg)
        eeg = self.layer10_eeg(eeg)
        tmp = self.layer11_eeg(eeg)
        eeg = tmp + eeg
        eeg = self.maxpool_eeg_layer11(eeg)
        tmp = self.layer12_eeg(eeg)
        eeg = tmp + eeg
        eeg = torch.flatten(eeg, 1)
        eeg = F.leaky_relu(self.linear_eeg(eeg))

        combined = F.leaky_relu(self.linear_combined1(torch.cat((ecg, art, eeg), -1))
        logits = self.linear_combined2(combined)
        # probs = F.sigmoid(logits)

        return logits


loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 3 min model
torch.manual_seed(seed)
model = model_leaky_relu()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_3min_leaky_relu.pth"
model = model_train(train_loader_3min, val_loader_3min, model, loss_func, optimiz
```

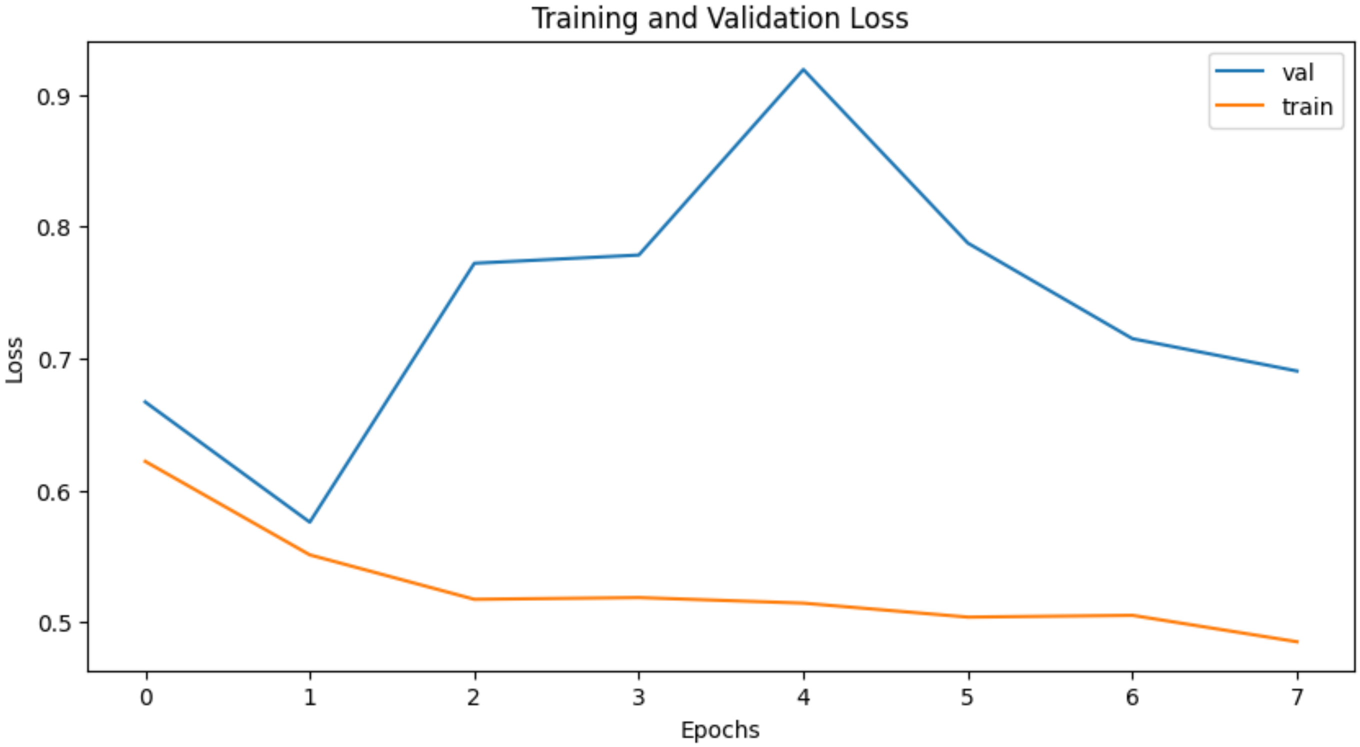
```
Epoch: 0, Train loss: 0.5850250395026837, Train accuracy: 0.6976893544197083
Val loss: 0.5358009242585727, Val accuracy: 0.698876404494382
Epoch: 1, Train loss: 0.5049068588423943, Train accuracy: 0.7496790885925293
Val loss: 0.5378439554146357, Val accuracy: 0.7393258426966293
Epoch: 2, Train loss: 0.45416892936814274, Train accuracy: 0.7848202586174011
Val loss: 0.5477120972105435, Val accuracy: 0.7370786516853932
Epoch: 3, Train loss: 0.4425018416366161, Train accuracy: 0.787708580493927
Val loss: 0.547105796635151, Val accuracy: 0.7404494382022472
Epoch: 4, Train loss: 0.42863916899334636, Train accuracy: 0.801508367061615
Val loss: 0.4621351435780526, Val accuracy: 0.7764044943820225
Epoch: 5, Train loss: 0.4223131165577909, Train accuracy: 0.8079268336296082
```

```
Epoch: 5, Train loss: 0.4225151165577909, Train accuracy: 0.8079208556296082
Val loss: 0.4707101743136133, Val accuracy: 0.7775280898876404
Epoch: 6, Train loss: 0.4041486335045566, Train accuracy: 0.8117779493331909
Val loss: 0.4660787784627506, Val accuracy: 0.7775280898876404
Epoch: 7, Train loss: 0.4009761328262603, Train accuracy: 0.8206033110618591
Val loss: 0.4433167204260826, Val accuracy: 0.7955056179775281
Epoch: 8, Train loss: 0.3892250312384653, Train accuracy: 0.8214056491851807
Val loss: 0.4498185664415359, Val accuracy: 0.7910112359550562
Epoch: 9, Train loss: 0.3813350093533047, Train accuracy: 0.8307124376296997
Val loss: 0.5158824090446745, Val accuracy: 0.7853932584269663
Epoch: 10, Train loss: 0.36878103576361443, Train accuracy: 0.833921670913696
Val loss: 0.4488084986805916, Val accuracy: 0.7966292134831461
Epoch: 11, Train loss: 0.3554885527441835, Train accuracy: 0.8424261808395386
Val loss: 0.4907365973506654, Val accuracy: 0.7719101123595505
Epoch: 12, Train loss: 0.34804952768214403, Train accuracy: 0.847079575061798
Val loss: 0.45259463042020803, Val accuracy: 0.7966292134831461
Epoch: 13, Train loss: 0.33242526319504395, Train accuracy: 0.863125801086425
Val loss: 0.4612286686897278, Val accuracy: 0.7865168539325843
Early stopped training at epoch 13
```



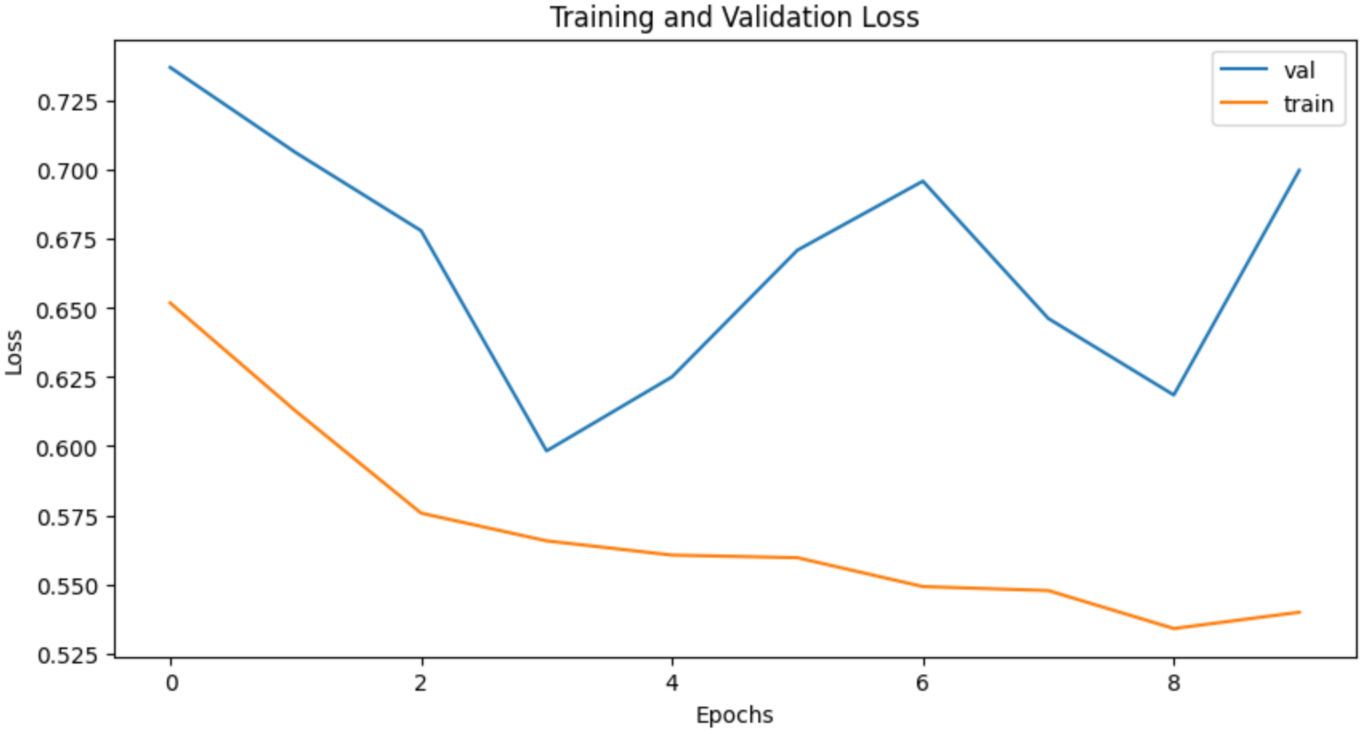
```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
```

```
early_stop_thresh = 5

# 5 min model
torch.manual_seed(seed)
model = model_leaky_relu()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_5min_leaky_relu.pth"
model = model_train(train_loader_5min, val_loader_5min, model, loss_func, optimiz
Epoch: 0, Train loss: 0.6219721039842961, Train accuracy: 0.6457558870315552
Val loss: 0.6669927494866508, Val accuracy: 0.579185520361991
Epoch: 1, Train loss: 0.5511174921750445, Train accuracy: 0.7081649303436279
Val loss: 0.5759000671761375, Val accuracy: 0.6843891402714932
Epoch: 2, Train loss: 0.5174144360454408, Train accuracy: 0.7299919128417969
Val loss: 0.7721297176820893, Val accuracy: 0.581447963800905
Epoch: 3, Train loss: 0.5187308560867124, Train accuracy: 0.7228779196739197
Val loss: 0.7783715927175112, Val accuracy: 0.582579185520362
Epoch: 4, Train loss: 0.5145076268270503, Train accuracy: 0.7324171662330627
Val loss: 0.9190180578402111, Val accuracy: 0.5
Epoch: 5, Train loss: 0.5039204737594654, Train accuracy: 0.7383993268013
Val loss: 0.7873900788170951, Val accuracy: 0.6063348416289592
Epoch: 6, Train loss: 0.5052736438949357, Train accuracy: 0.7408245801925659
Val loss: 0.7149691166622298, Val accuracy: 0.6052036199095022
Epoch: 7, Train loss: 0.48524061195678836, Train accuracy: 0.751010537147522
Val loss: 0.6904633939266205, Val accuracy: 0.6176470588235294
Early stopped training at epoch 7
```



```
loss_func = nn.BCEWithLogitsLoss()
lr = 0.001
num_epoch = 100
early_stop_thresh = 5
```

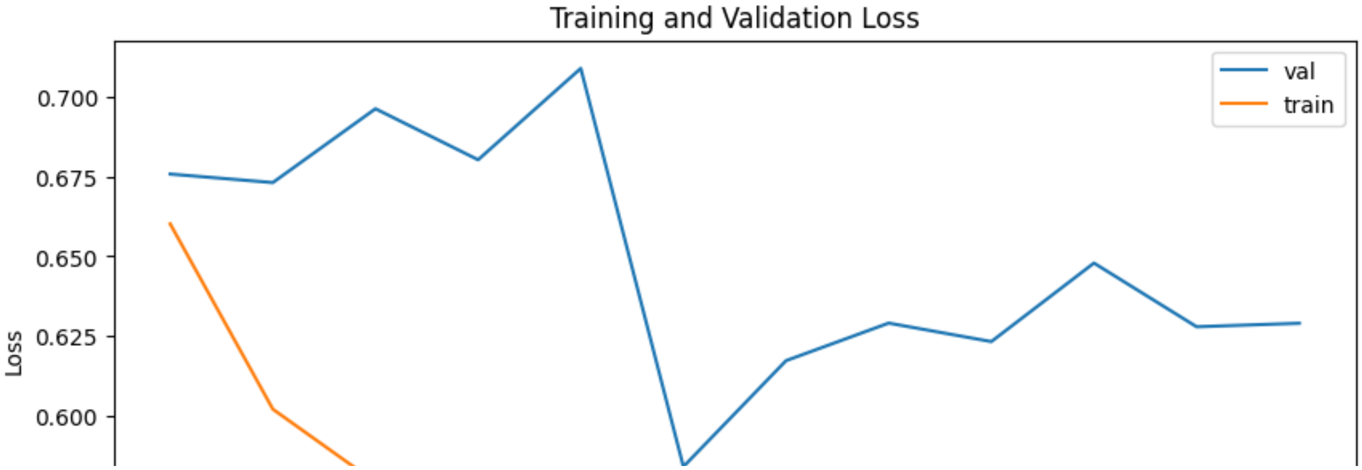
```
# 10 min model
torch.manual_seed(seed)
model = model_leaky_relu()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_10min_leaky_relu.pth
model = model_train(train_loader_10min, val_loader_10min, model, loss_func, optim
Epoch: 0, Train loss: 0.6519051532344062, Train accuracy: 0.6244224309921265
Val loss: 0.7372486591339111, Val accuracy: 0.4237875288683603
Epoch: 1, Train loss: 0.6127323769696869, Train accuracy: 0.656105637550354
Val loss: 0.7063896038702557, Val accuracy: 0.4653579676674365
Epoch: 2, Train loss: 0.575741438149619, Train accuracy: 0.6735973358154297
Val loss: 0.6780814113361494, Val accuracy: 0.6316397228637414
Epoch: 3, Train loss: 0.5656948842624626, Train accuracy: 0.6867986917495728
Val loss: 0.598285725074155, Val accuracy: 0.6385681293302541
Epoch: 4, Train loss: 0.5605388104325474, Train accuracy: 0.686963677406311
Val loss: 0.6251096480659075, Val accuracy: 0.6108545034642032
Epoch: 5, Train loss: 0.5595799359551358, Train accuracy: 0.6930692791938782
Val loss: 0.6710381891046251, Val accuracy: 0.6108545034642032
Epoch: 6, Train loss: 0.5491262307851621, Train accuracy: 0.698019802570343
Val loss: 0.6960419906037193, Val accuracy: 0.5808314087759815
Epoch: 7, Train loss: 0.5476787087744218, Train accuracy: 0.6998350024223328
Val loss: 0.6462656302111489, Val accuracy: 0.5715935334872979
Epoch: 8, Train loss: 0.5339273803304918, Train accuracy: 0.7125412821769714
Val loss: 0.6185505262442995, Val accuracy: 0.5958429561200924
Epoch: 9, Train loss: 0.5398287240034676, Train accuracy: 0.7115511298179626
Val loss: 0.699994829084192, Val accuracy: 0.5184757505773672
Early stopped training at epoch 9
```

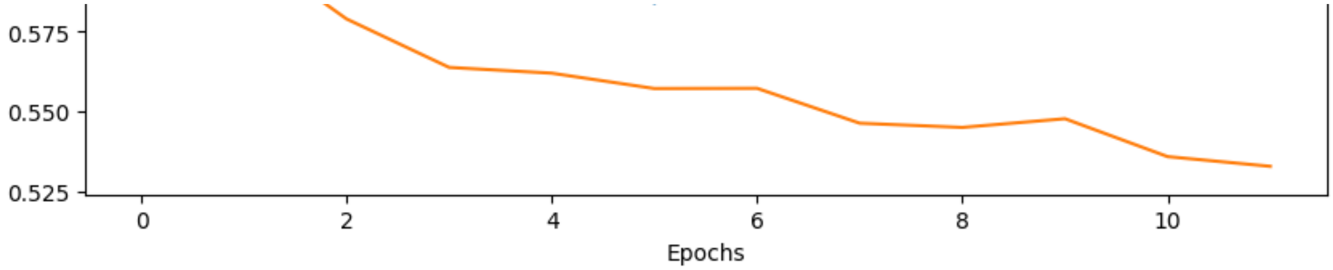


```
loss_func = nn.BCEWithLogitsLoss()
```

```
lr = 0.001
num_epoch = 100
early_stop_thresh = 5

# 15 min model
torch.manual_seed(seed)
model = model_leaky_relu()
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
best_model_path = "/content/drive/MyDrive/VitalDB/best_model_15min_leaky_relu.pth
model = model_train(train_loader_15min, val_loader_15min, model, loss_func, optim
```

```
Epoch: 0, Train loss: 0.6601643356016057, Train accuracy: 0.5971990823745728
Val loss: 0.6757981975873312, Val accuracy: 0.6382730455075846
Epoch: 1, Train loss: 0.6020179858959767, Train accuracy: 0.6512170433998108
Val loss: 0.6731257129598547, Val accuracy: 0.5530921820303384
Epoch: 2, Train loss: 0.5790417654468998, Train accuracy: 0.6777259111404419
Val loss: 0.6962931045779476, Val accuracy: 0.5787631271878646
Epoch: 3, Train loss: 0.5638474392549083, Train accuracy: 0.690396785736084
Val loss: 0.6802825033664702, Val accuracy: 0.544924154025671
Epoch: 4, Train loss: 0.5620372569056183, Train accuracy: 0.6843947768211365
Val loss: 0.7089544159394726, Val accuracy: 0.5157526254375729
Epoch: 5, Train loss: 0.557225272173721, Train accuracy: 0.6832277178764343
Val loss: 0.5839641105245662, Val accuracy: 0.6499416569428238
Epoch: 6, Train loss: 0.5572938340113297, Train accuracy: 0.6872290968894958
Val loss: 0.6172353205857454, Val accuracy: 0.6266044340723453
Epoch: 7, Train loss: 0.5463678442605061, Train accuracy: 0.6968989372253418
Val loss: 0.6290234172785725, Val accuracy: 0.6429404900816803
Epoch: 8, Train loss: 0.5450468476154916, Train accuracy: 0.694231390953064
Val loss: 0.6232225563791063, Val accuracy: 0.6697782963827305
Epoch: 9, Train loss: 0.5477718500722921, Train accuracy: 0.6910637021064758
Val loss: 0.647865714850249, Val accuracy: 0.5787631271878646
Epoch: 10, Train loss: 0.5358982775838584, Train accuracy: 0.7044014930725098
Val loss: 0.6278779009977976, Val accuracy: 0.6219369894982497
Epoch: 11, Train loss: 0.5328560442716211, Train accuracy: 0.7010670304298401
Val loss: 0.6289723338904204, Val accuracy: 0.6382730455075846
Early stopped training at epoch 11
```





# Results

In this section, you should finish training your model training or loading your trained model. That is a great experiment! You should share the results with others with necessary metrics and figures.

Please test and report results for all experiments that you run with:

- specific numbers (accuracy, AUC, RMSE, etc)
- figures (loss shrinkage, outputs from GAN, annotation or label of sample pictures, etc)

# Evaluation

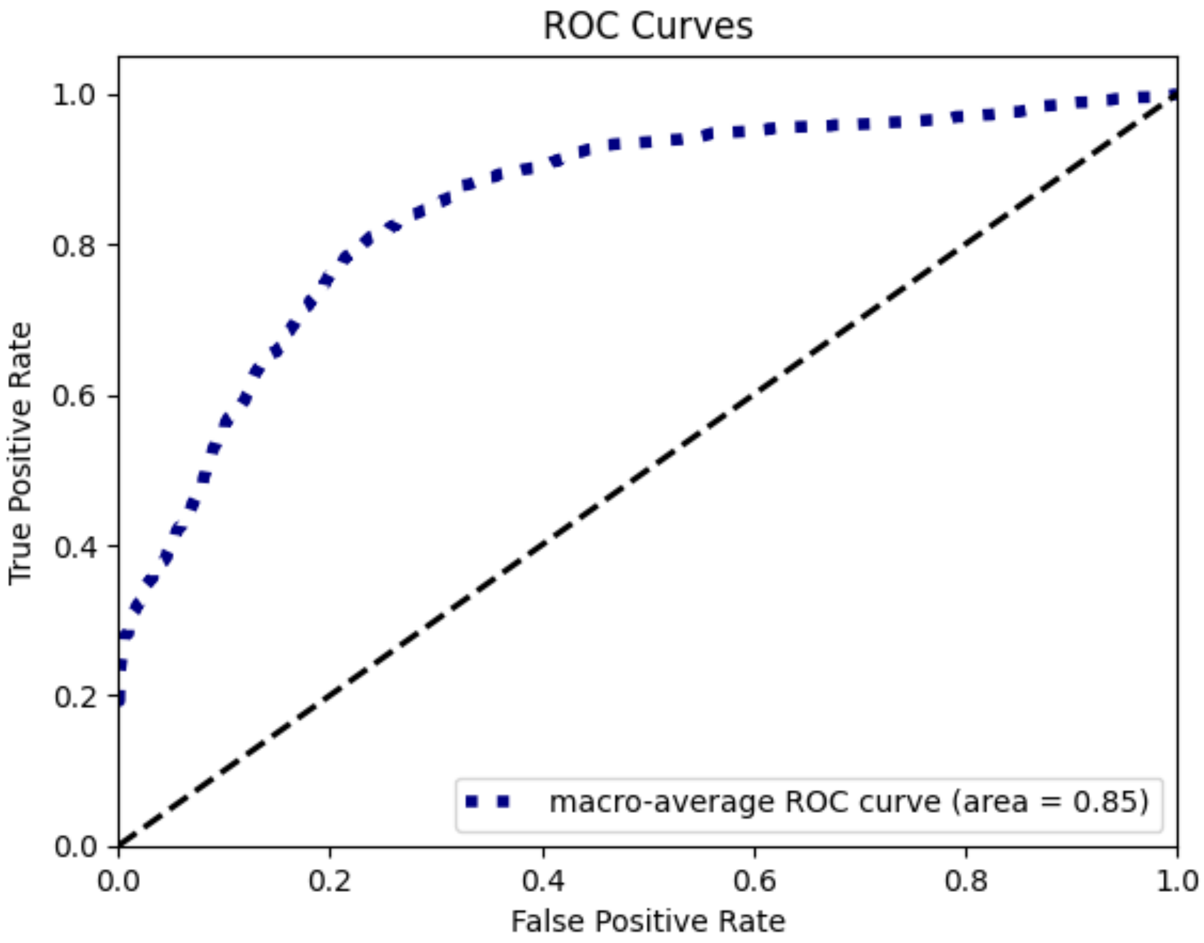
```
def eval_model(test_dataloader, model):

    model.eval()
    Y_score = []
    Y_pred = []
    Y_true = []

    for (ecg, art, eeg), y in test_dataloader:

        y_hat = torch.sigmoid(model(ecg, art, eeg))
        y_hat = y_hat.detach()
        Y_score.append(y_hat)

        y_hat = (y_hat>0.5).int()
        Y_pred.append(y_hat)

        Y_true.append(y)

    Y_score = np.concatenate(Y_score, axis=0)
    Y_pred = np.concatenate(Y_pred, axis=0)
    Y_true = np.concatenate(Y_true, axis=0)

    return Y_score, Y_pred, Y_true
```

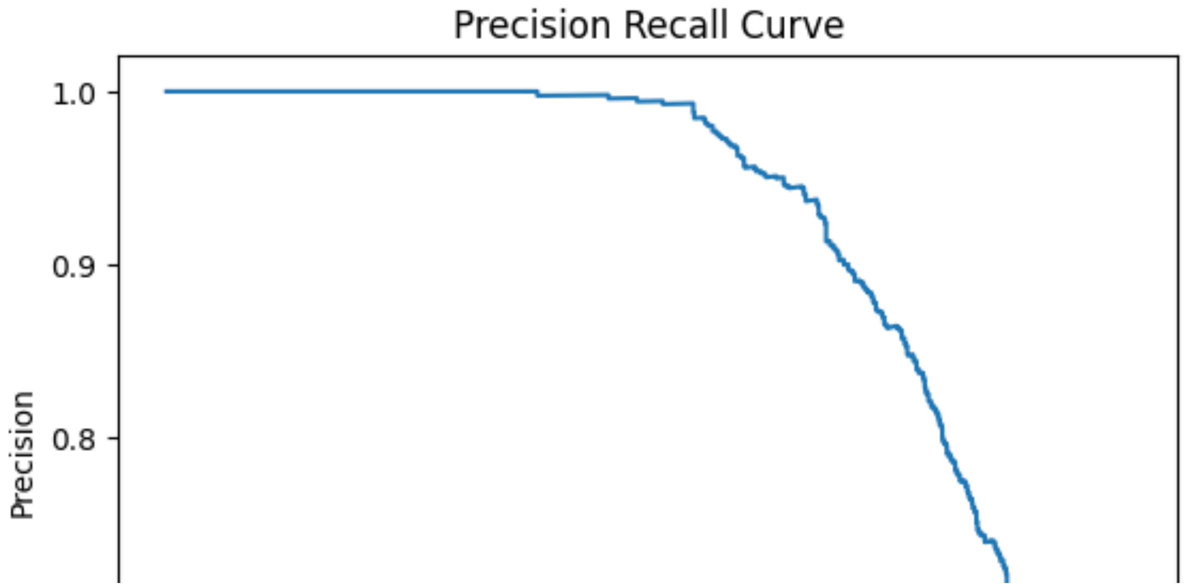
```
def print_eval_parameters(test_dataloader, model_architecture, lr, optimizer, PAT

  model = model_architecture
  lr = lr
  optimizer = optimizer(model.parameters(), lr = lr)

  resume(model, optimizer, PATH)

  y_score, y_pred, y_true = eval_model(test_dataloader, model)

  # calculate result metrics
  acc = accuracy_score(y_true, y_pred)
  roc_auc = roc_auc_score(y_true, y_score)
  pr_auc = average_precision_score(y_true, y_score)

  print(f'Test Accuracy: {acc}')
  print(f'Test AUC: {roc_auc}')
  print(f'Test PR AUC: {pr_auc}')

  # prepare score array for plotting
  y_score_oth_class = np.ones_like(y_score.T[0]) - y_score.T[0]
  y_score_comb = np.stack((y_score_oth_class, y_score.T[0]), axis=1)

  # Plot AUC
  skplt.plotters.plot_roc_curve(y_true, y_score_comb, curves=['macro'])
  plt.show()

  # Plot precision - recall curve
  precision, recall, thresholds = precision_recall_curve(y_true, y_score)
  plt.plot(recall, precision)
  plt.title('Precision Recall Curve')
  plt.xlabel('Recall')
  plt.ylabel('Precision')
  plt.show()

  return acc, roc_auc, pr_auc, y_true, y_score_comb
```

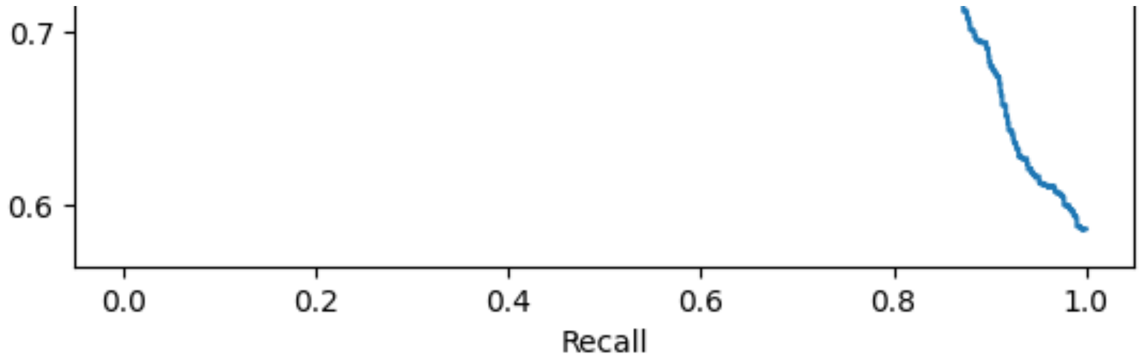
## Baseline model with best performance (batch=32 and lr=0.001)

```
# 3 minute model
acc_3min_batch32_lr001, roc_auc_3min_batch32_lr001, pr_auc_3min_batch32_lr001, y_
print_eval_parameters(test_loader_3min, my_model(), 0.001, torch.optim.Adam,
"/content/drive/MyDrive/VitalDB/best_model_3min_batch_32_lr
```

```
Test Accuracy: 0.7821448624368332
Test AUC: 0.8515936513877842
Test PR AUC: 0.91530439905873
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

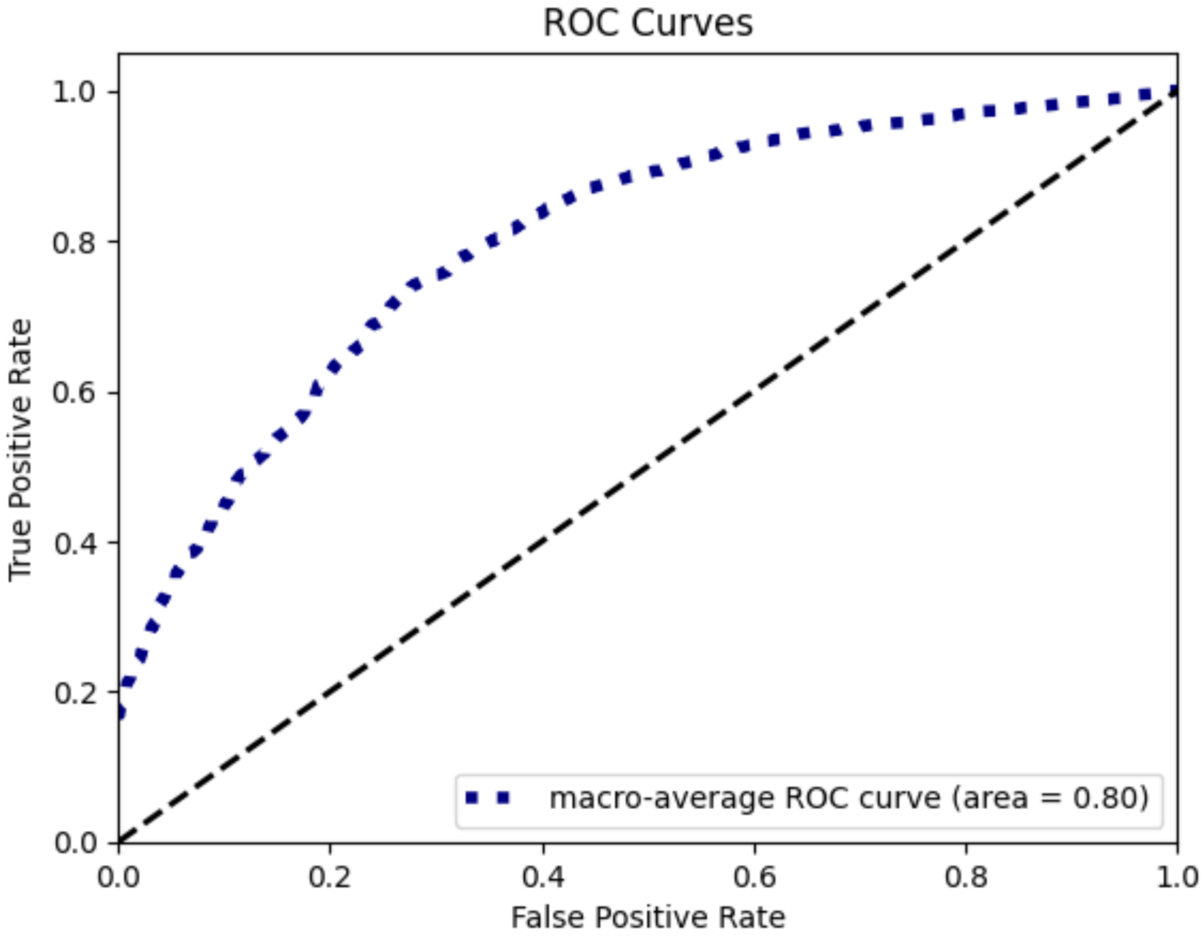


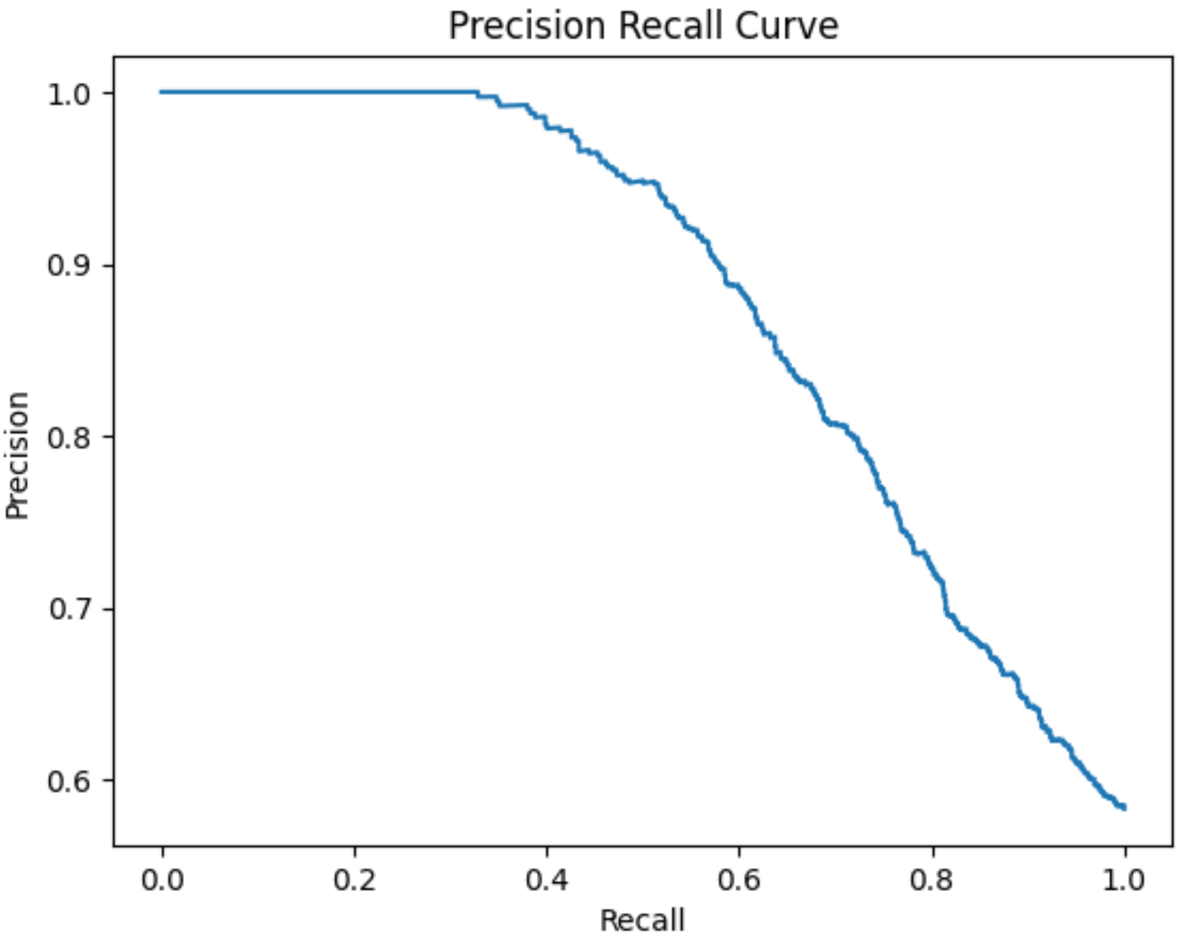




```
# 5 minute mo
acc_5min_batch32_lr001, roc_auc_5min_batch32_lr001, pr_auc_5min_batch32_lr001, y_
print_eval_parameters(test_loader_5min, my_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_5min_batch_32_lr
```

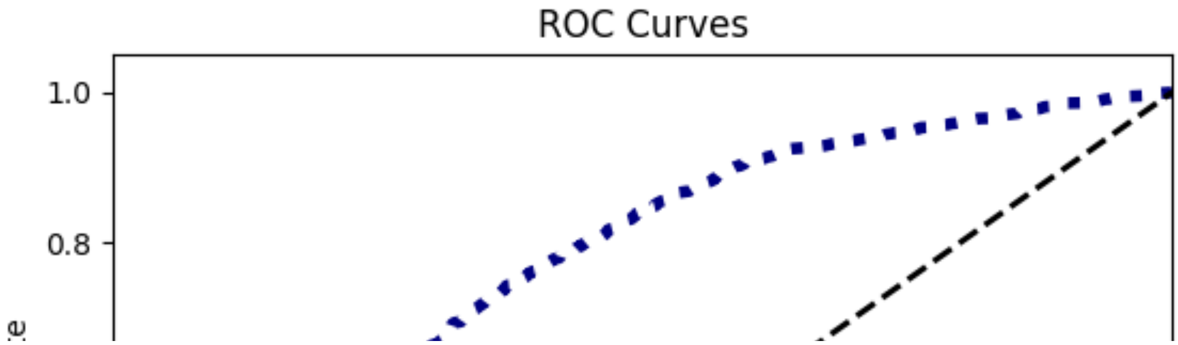
```
Test Accuracy: 0.6544117647058824
Test AUC: 0.8003788920664292
Test PR AUC: 0.8781751608813761
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

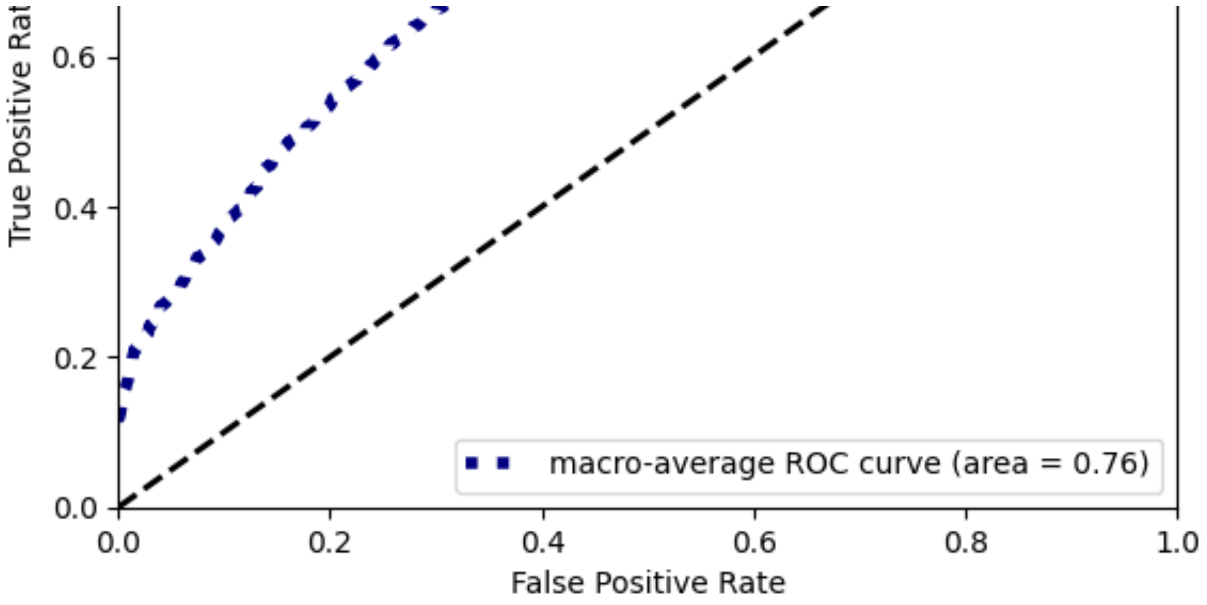


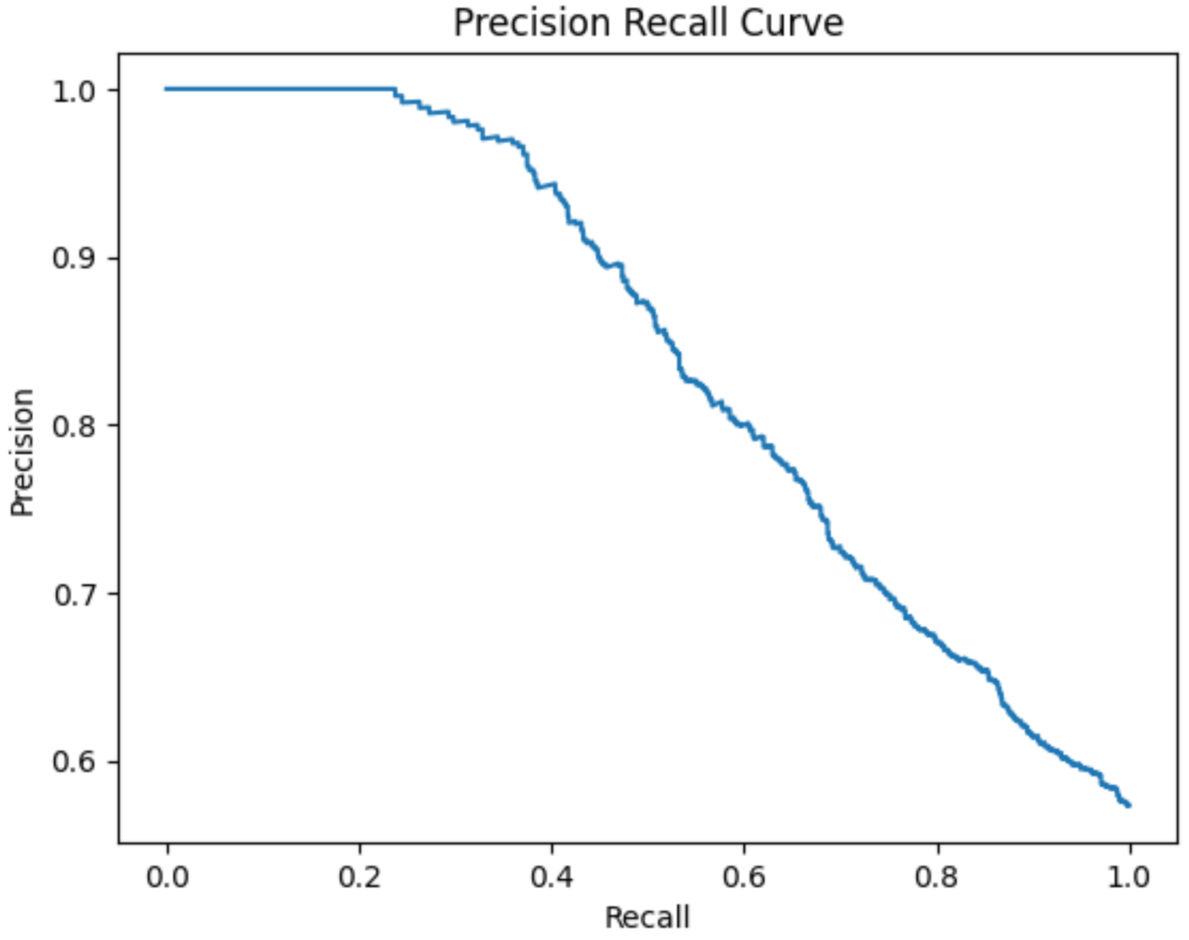


```
# 10 minute model
acc_10min_batch32_lr001, roc_auc_10min_batch32_lr001, pr_auc_10min_batch32_lr001,
print_eval_parameters(test_loader_10min, my_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_10min_batch_32_l
```

```
Test Accuracy: 0.6737875288683602
Test AUC: 0.756046043549774
Test PR AUC: 0.8400274429118524
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

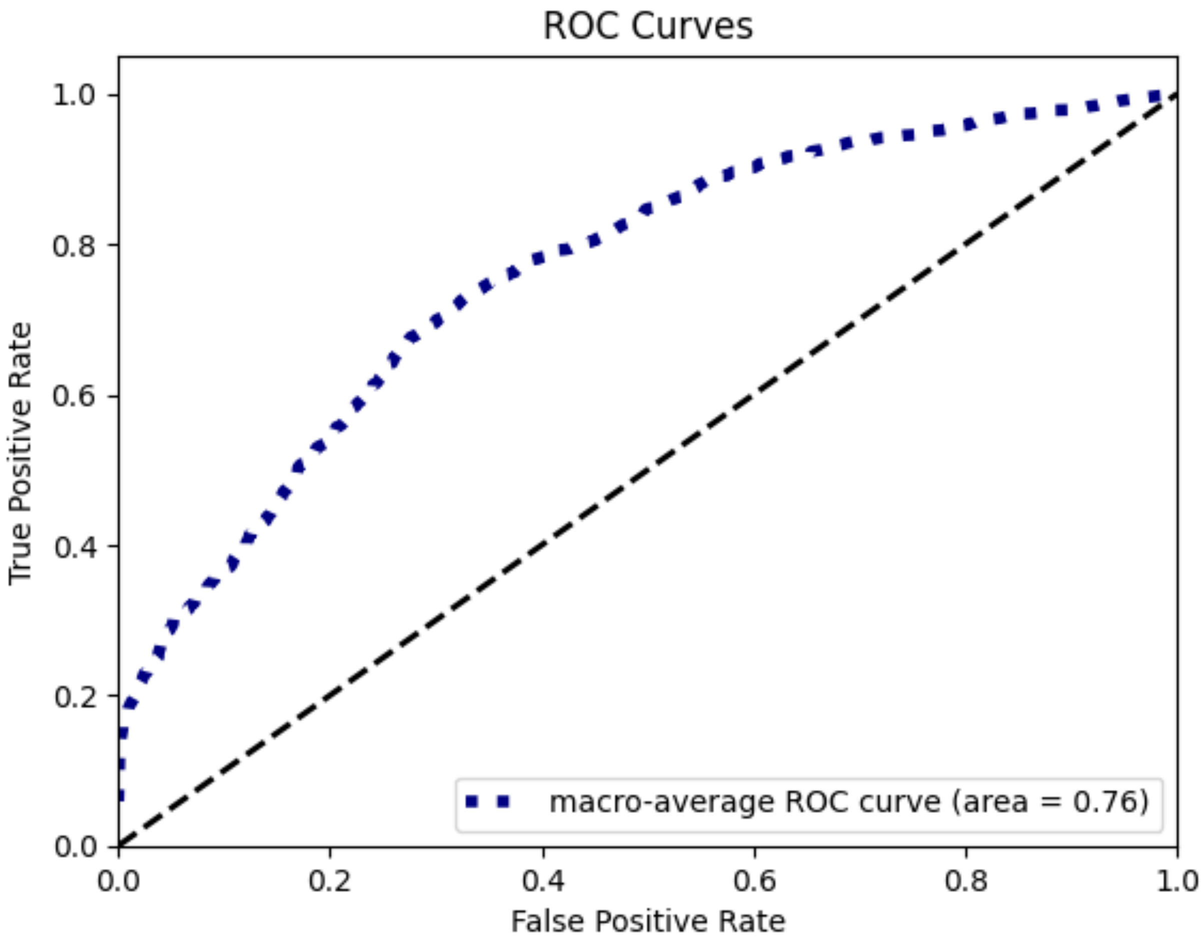


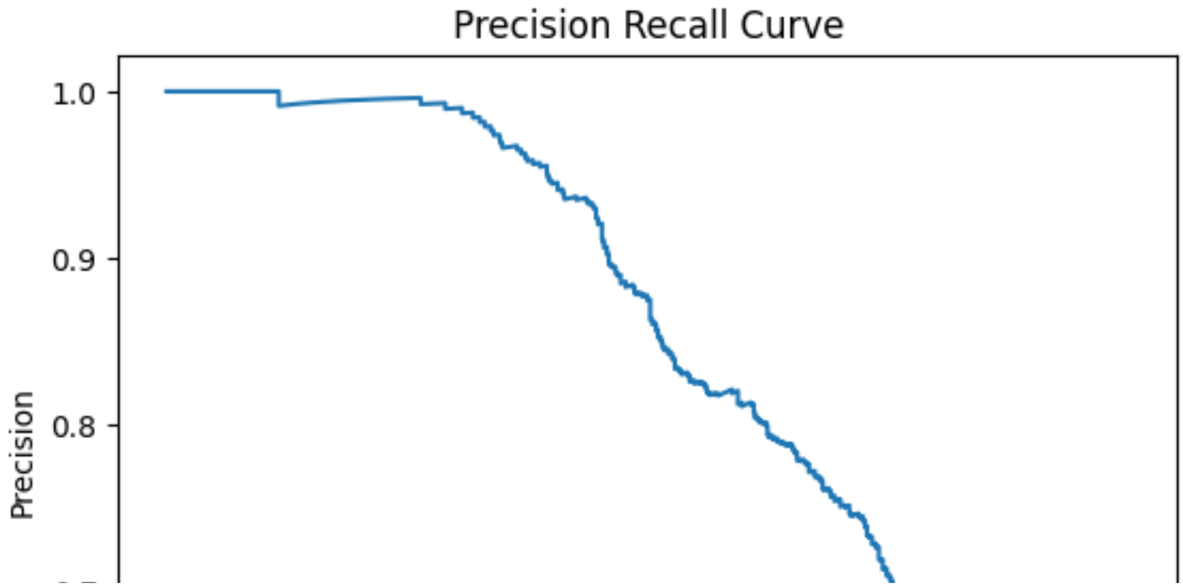


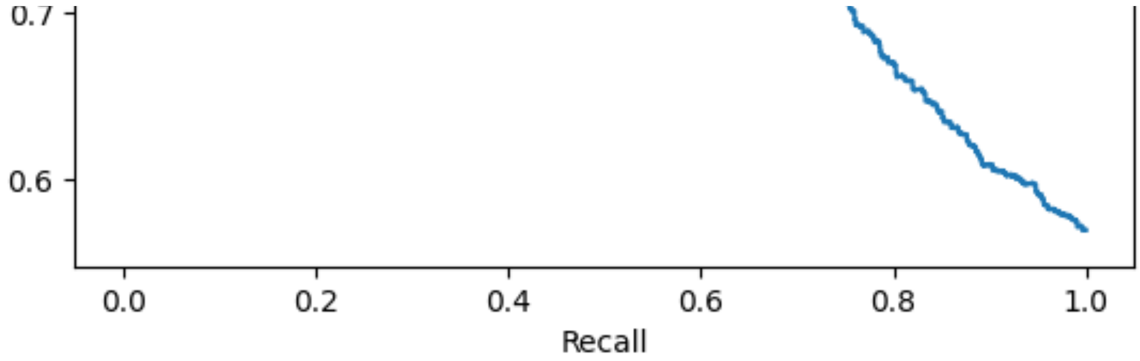


```
# 15 minute model
acc_15min_batch32_lr001, roc_auc_15min_batch32_lr001, pr_auc_15min_batch32_lr001,
print_eval_parameters(test_loader_15min, my_model(), 0.001, torch.optim.Adam,
"/content/drive/MyDrive/VitalDB/best_model_15min_batch_32_l
```

```
Test Accuracy: 0.6715285880980163
Test AUC: 0.7613440196431076
Test PR AUC: 0.8429138622019261
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```





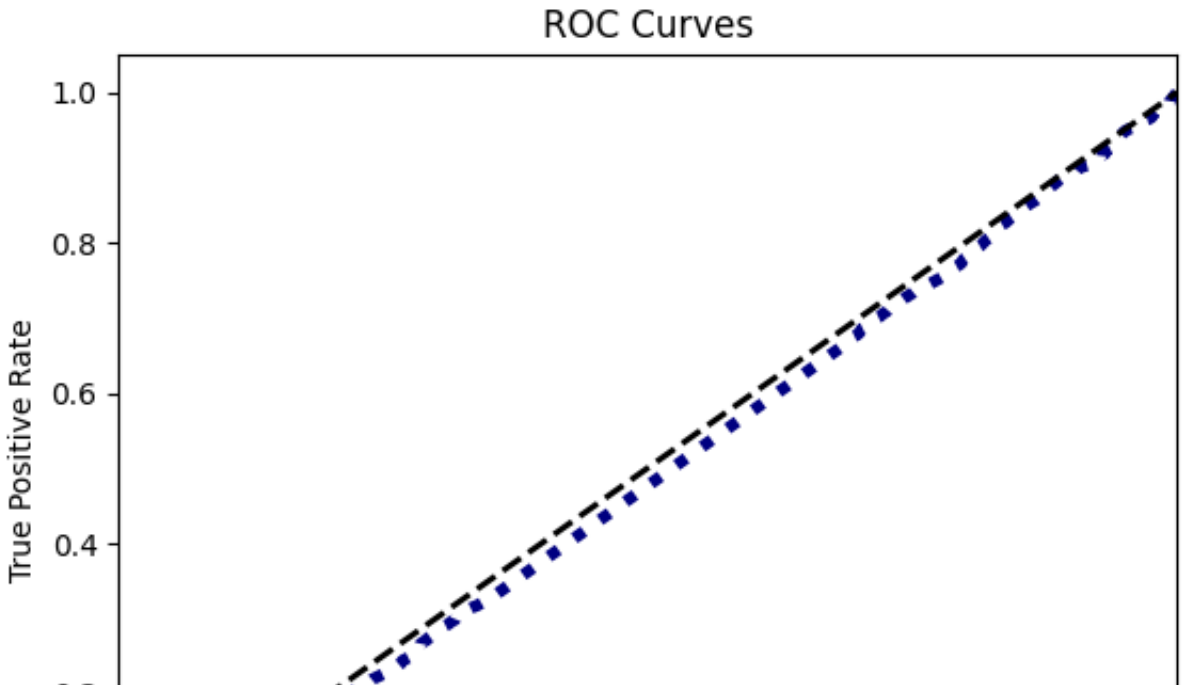


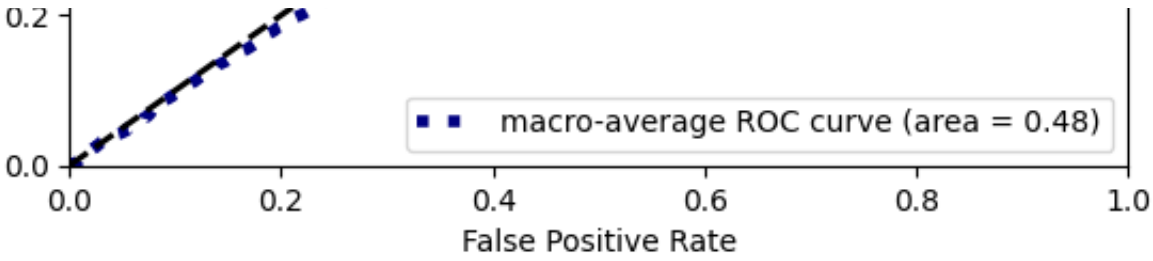
## Test effect of ablations

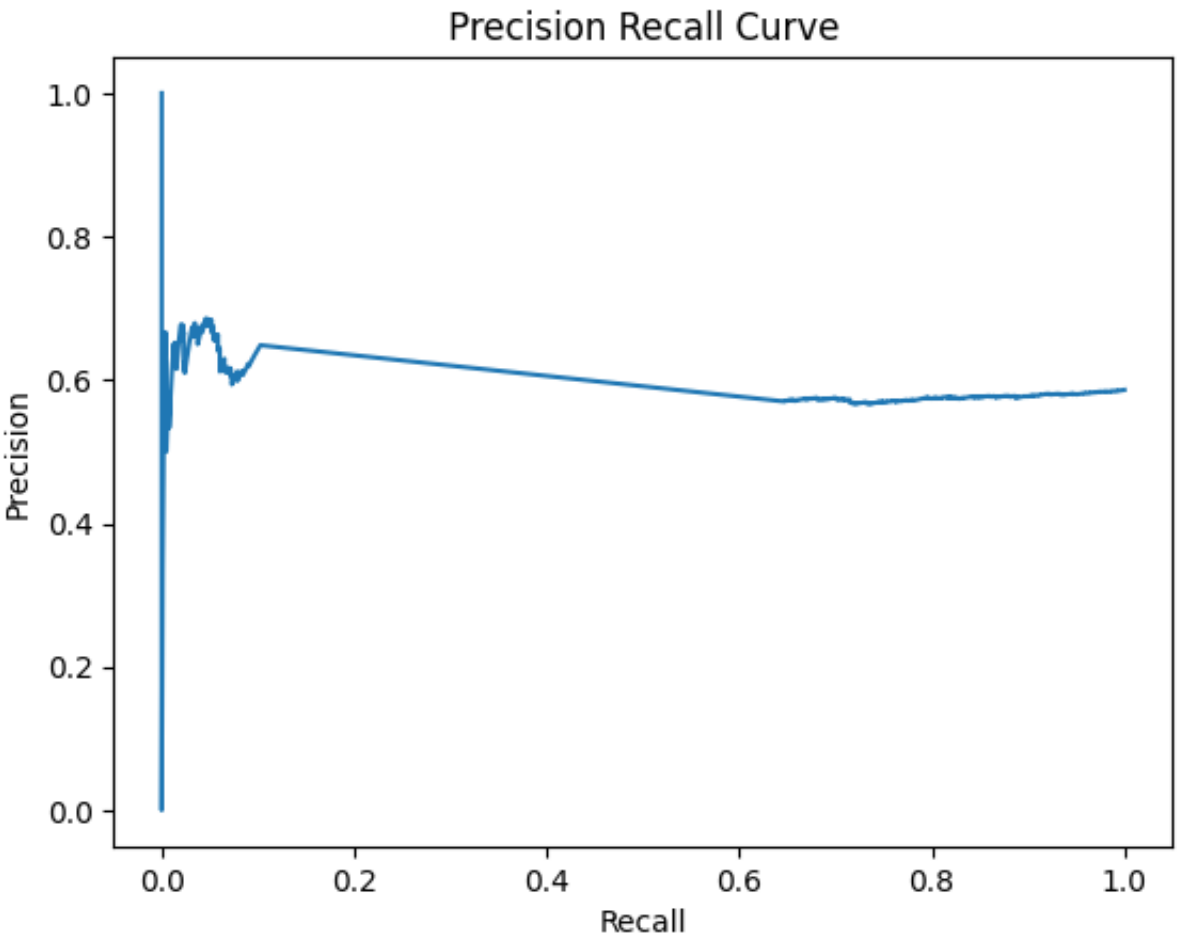
### ECG only

```
# 3 minute model
acc_3min_ecg, roc_auc_3min_ecg, pr_auc_3min_ecg, y_true_3min_ecg, y_score_comb_3m
print_eval_parameters(test_loader_3min, ecg_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_3min_ecg.pth")
```

```
Test Accuracy: 0.41493542953396967
Test AUC: 0.4831524197195839
Test PR AUC: 0.5787784263426045
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

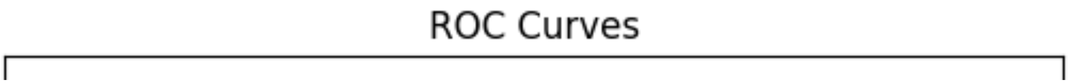


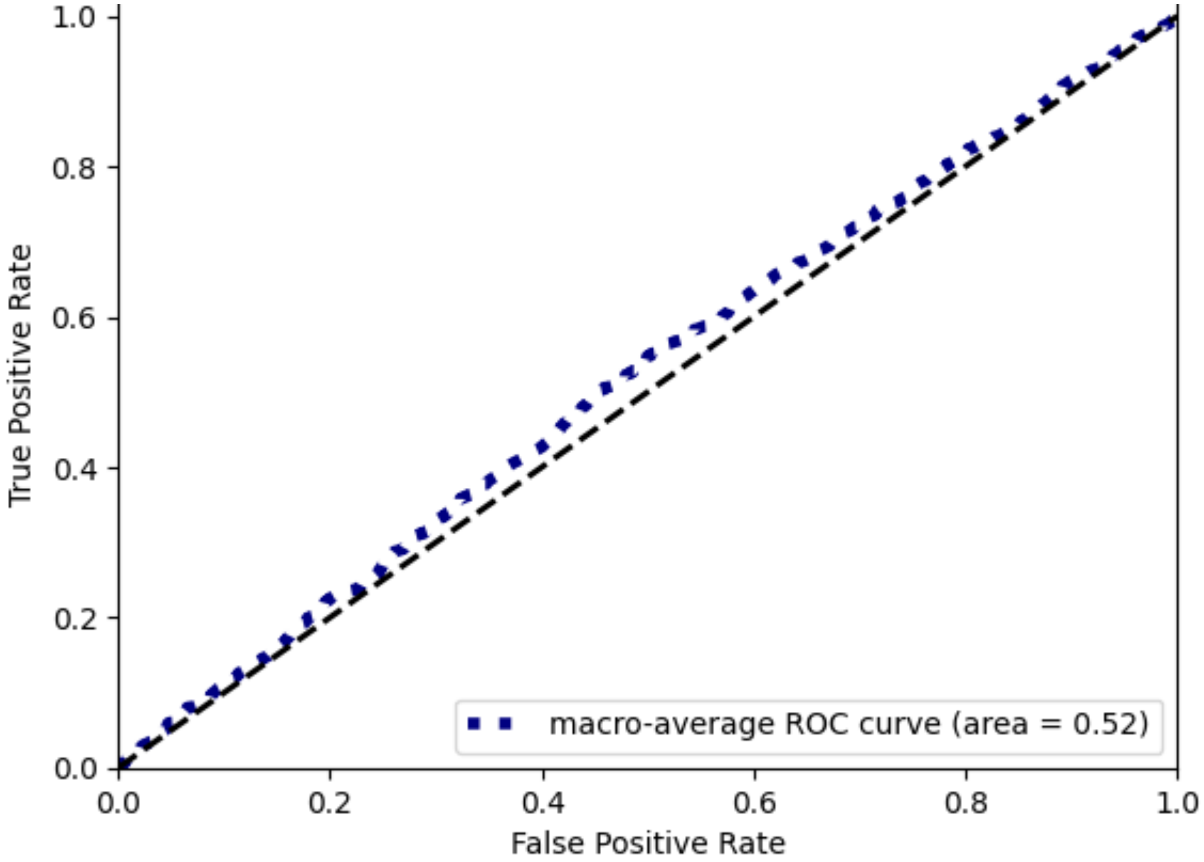


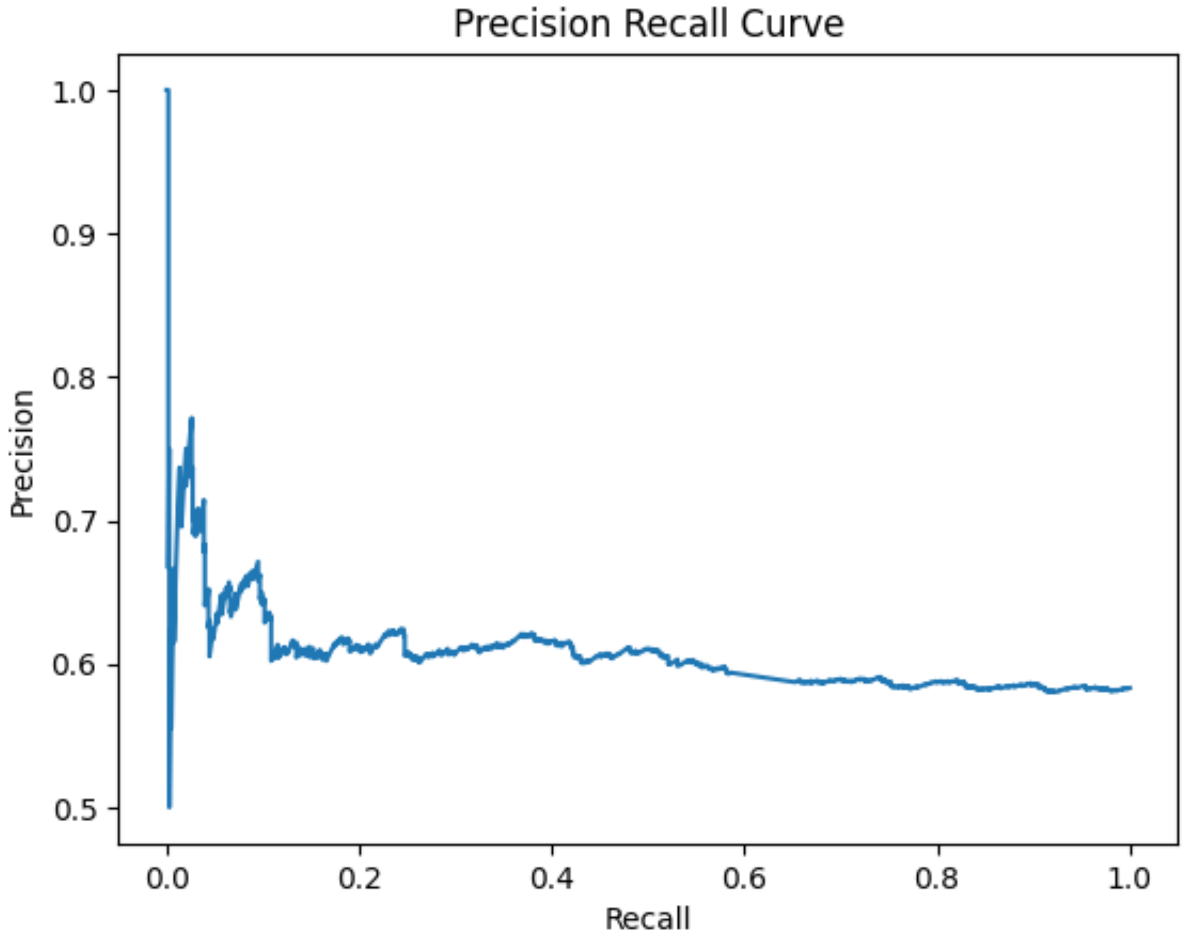


```
# 5 minute model
acc_5min_ecg, roc_auc_5min_ecg, pr_auc_5min_ecg, y_true_5min_ecg, y_score_comb_5m
print_eval_parameters(test_loader_5min, ecg_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_5min_ecg.pth")
```

```
Test Accuracy: 0.5831447963800905
Test AUC: 0.5209647468503528
Test PR AUC: 0.6064709050631483
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

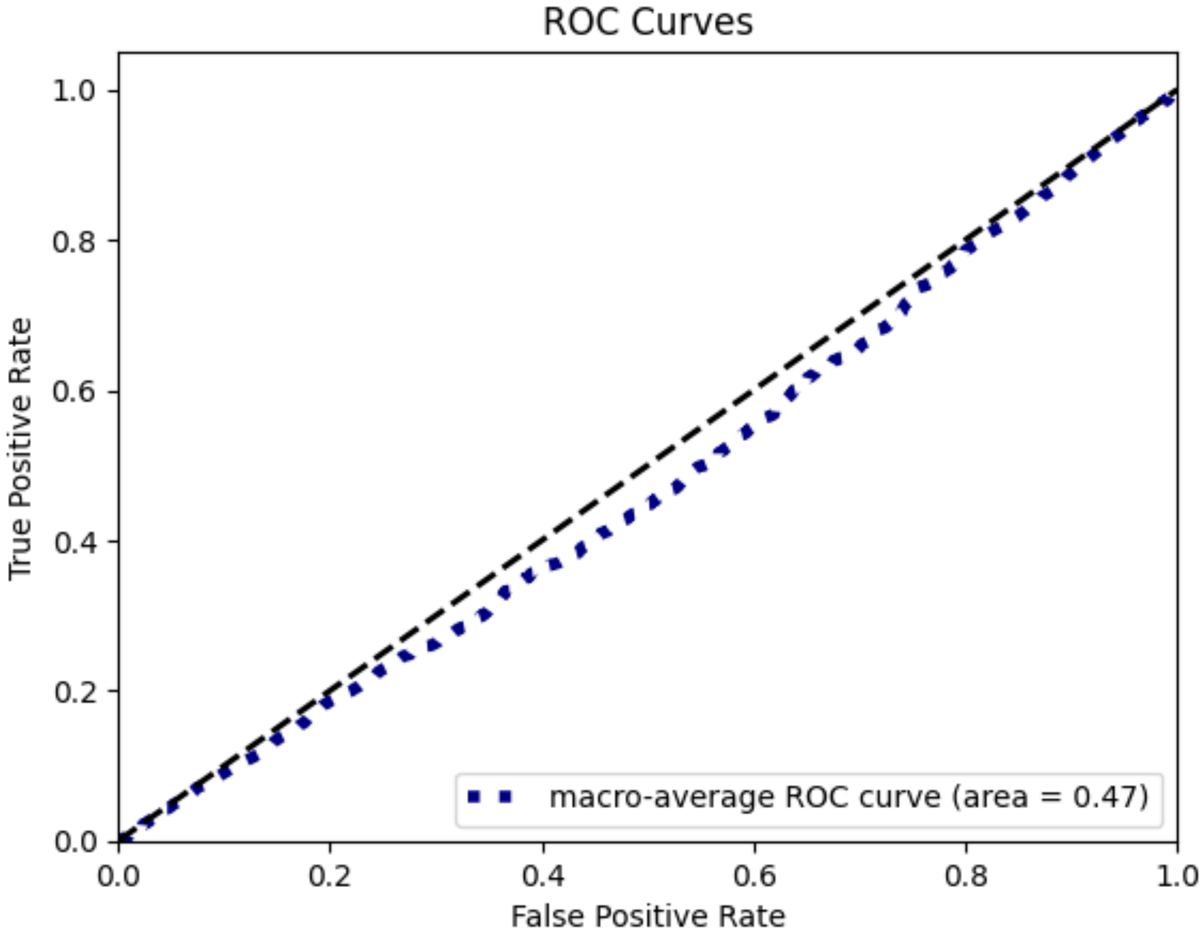


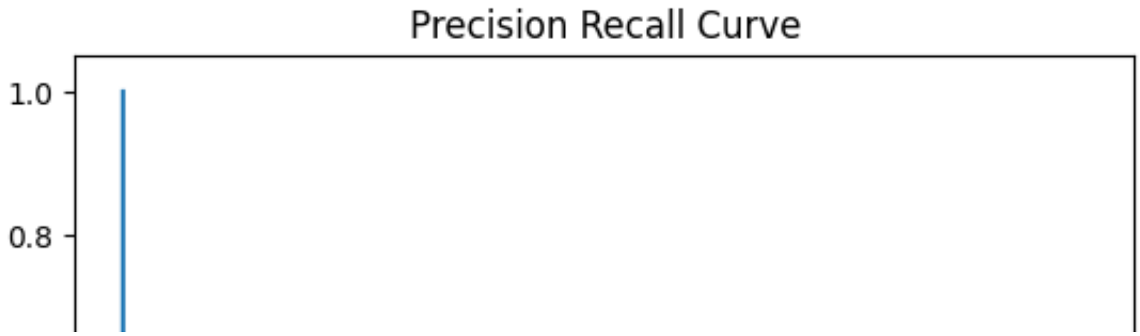

True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
macro-average ROC curve (area = 0.52)

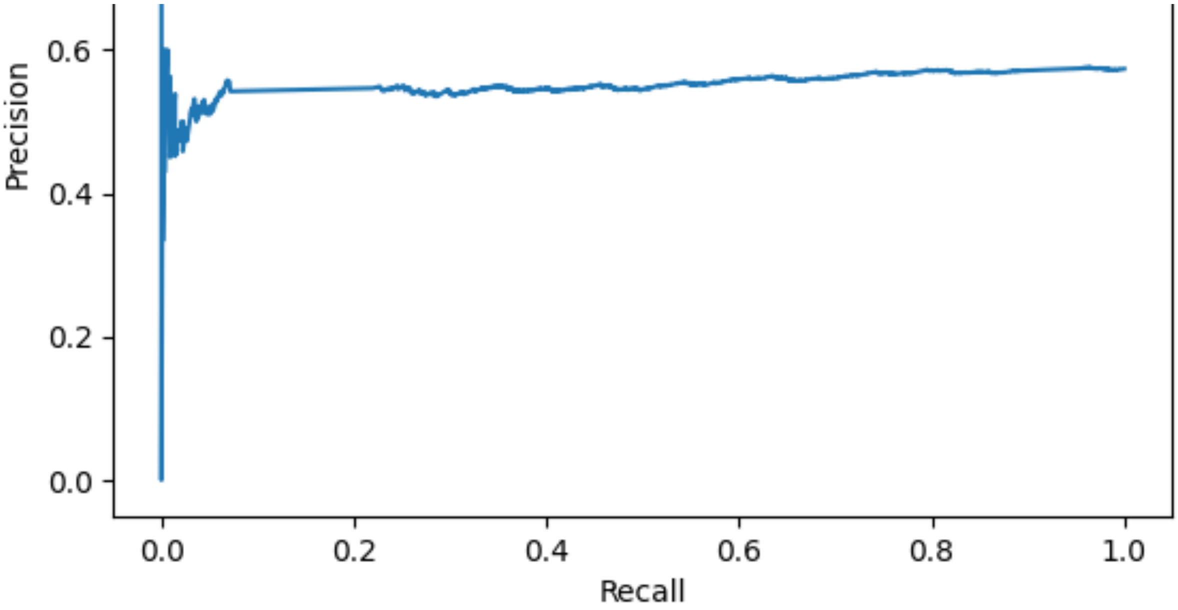

Precision Recall Curve
Precision
Recall
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.2
0.4
0.6
0.8
1.0

```
# 10 minute model
acc_10min_ecg, roc_auc_10min_ecg, pr_auc_10min_ecg, y_true_10min_ecg, y_score_com
print_eval_parameters(test_loader_10min, ecg_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_10min_ecg.pth")
```

```
Test Accuracy: 0.569284064665127
Test AUC: 0.47216033206736735
Test PR AUC: 0.5531092043744567
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

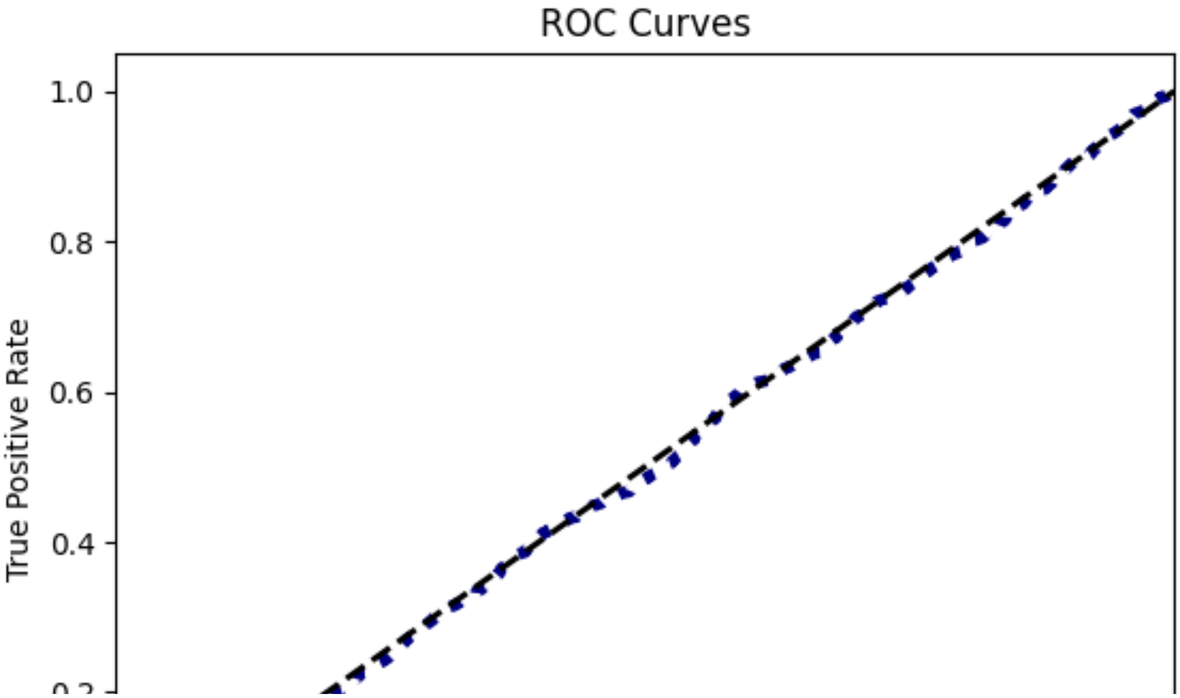


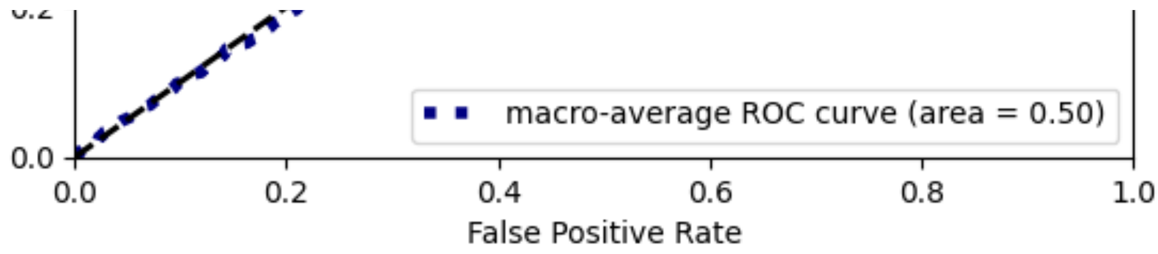


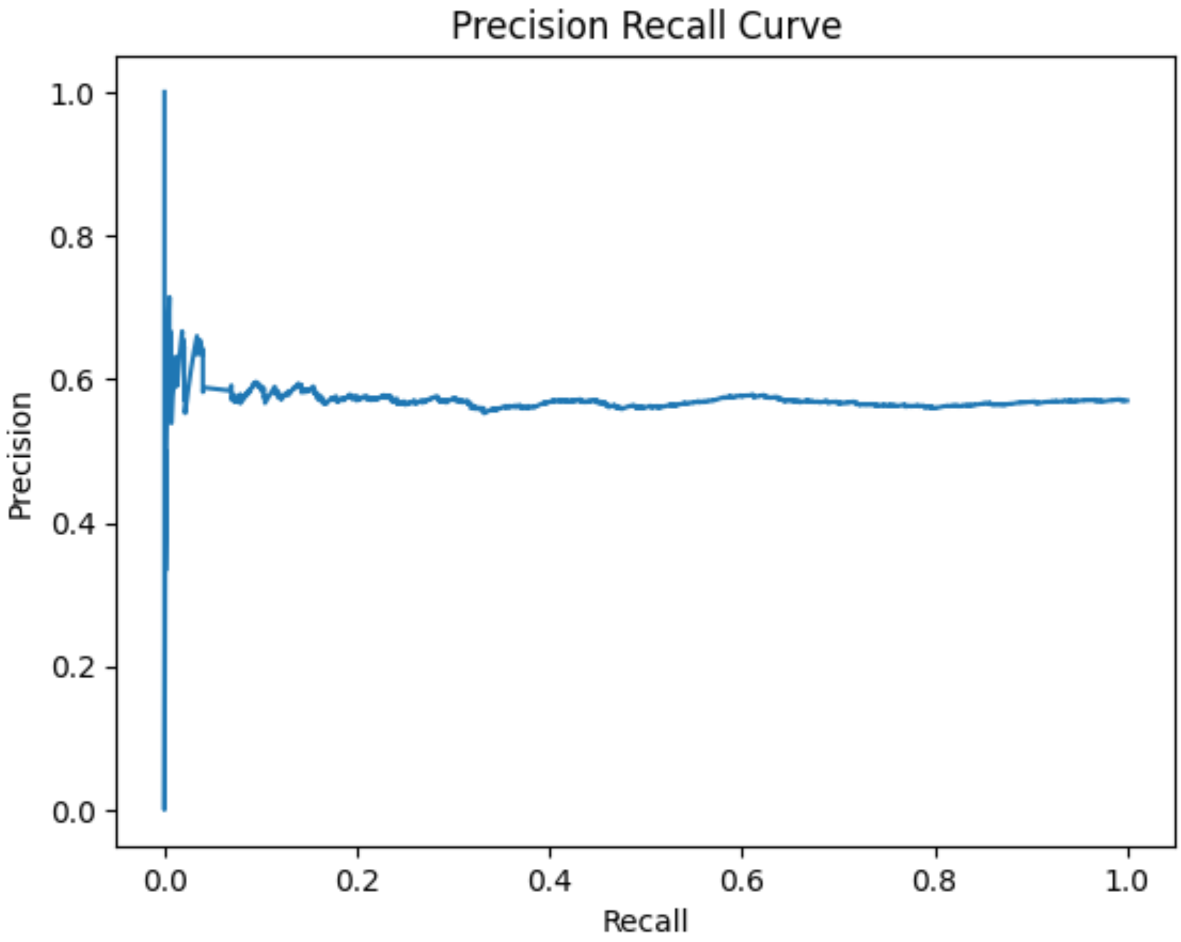


```
# 15 minute model
acc_3min_ecg, roc_auc_15min_ecg, pr_auc_15min_ecg, y_true_15min_ecg, y_score_comb
print_eval_parameters(test_loader_15min, ecg_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_15min_ecg.pth")
```

```
Test Accuracy: 0.5700116686114353
Test AUC: 0.4949552044374758
Test PR AUC: 0.5716493837622417
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```



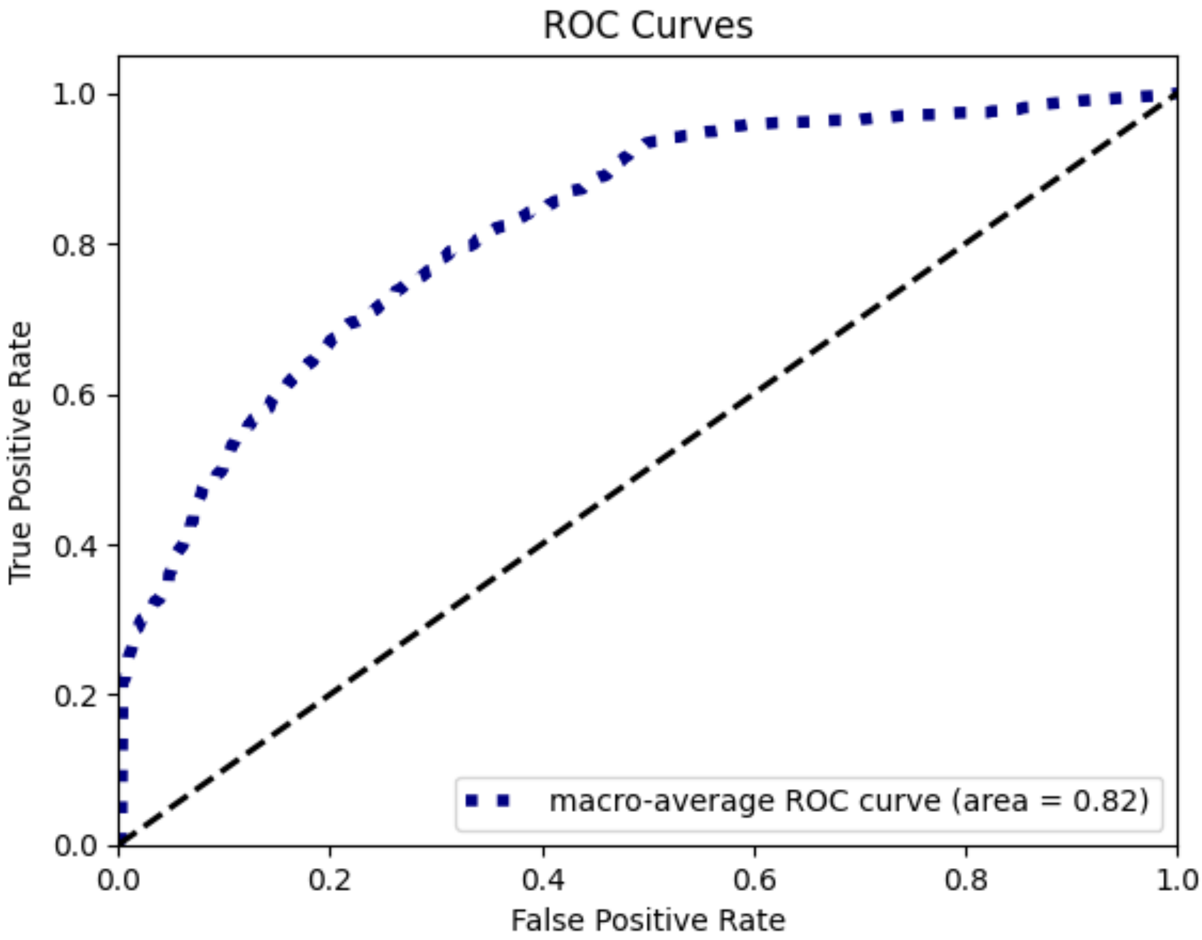


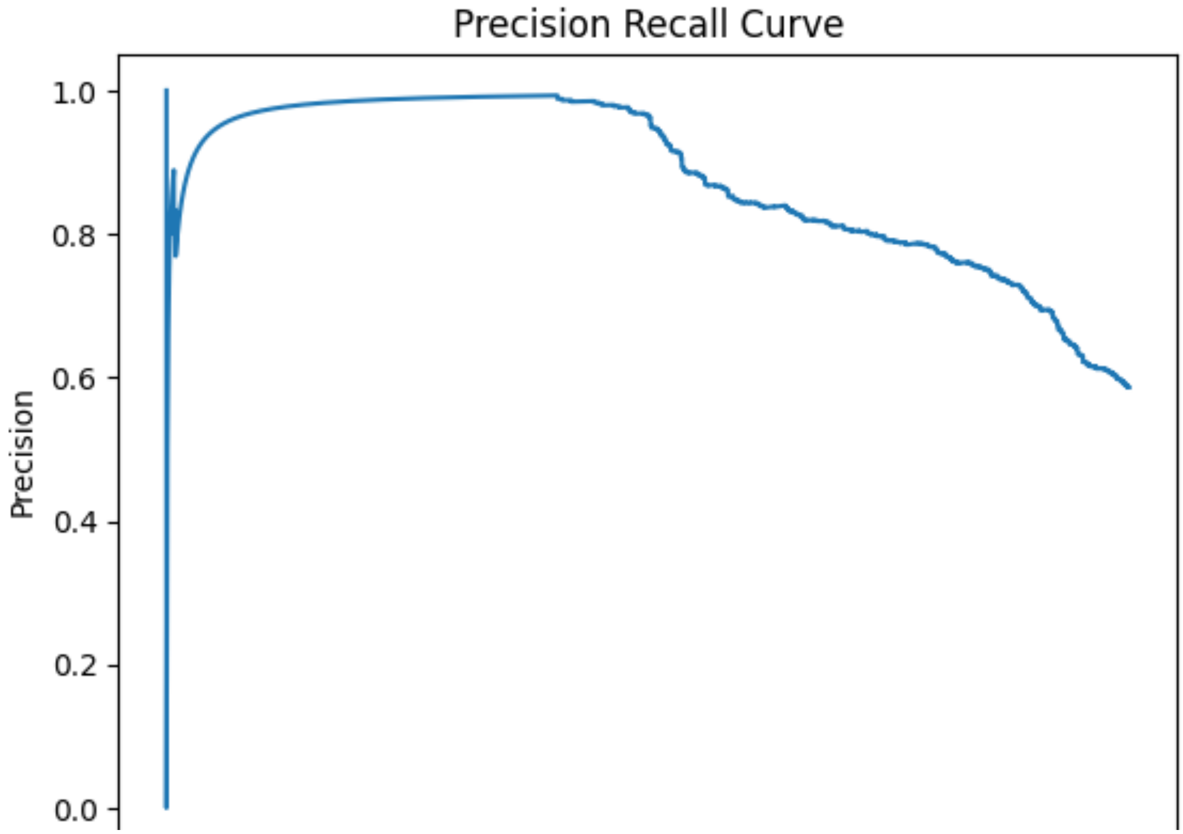


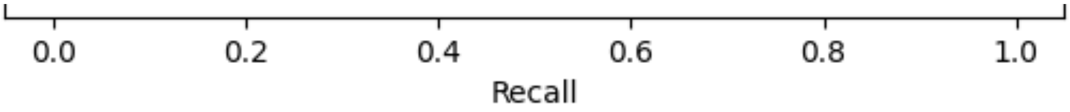
## ART model

```
# 3 minute model
acc_3min_art, roc_auc_3min_art, pr_auc_3min_art, y_true_3min_art, y_score_comb_3m
print_eval_parameters(test_loader_3min, art_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_3min_art.pth")
```

```
Test Accuracy: 0.6939921392476137
Test AUC: 0.8228625680375552
Test PR AUC: 0.8762367600200074
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
  warnings.warn(msg, category=FutureWarning)
```

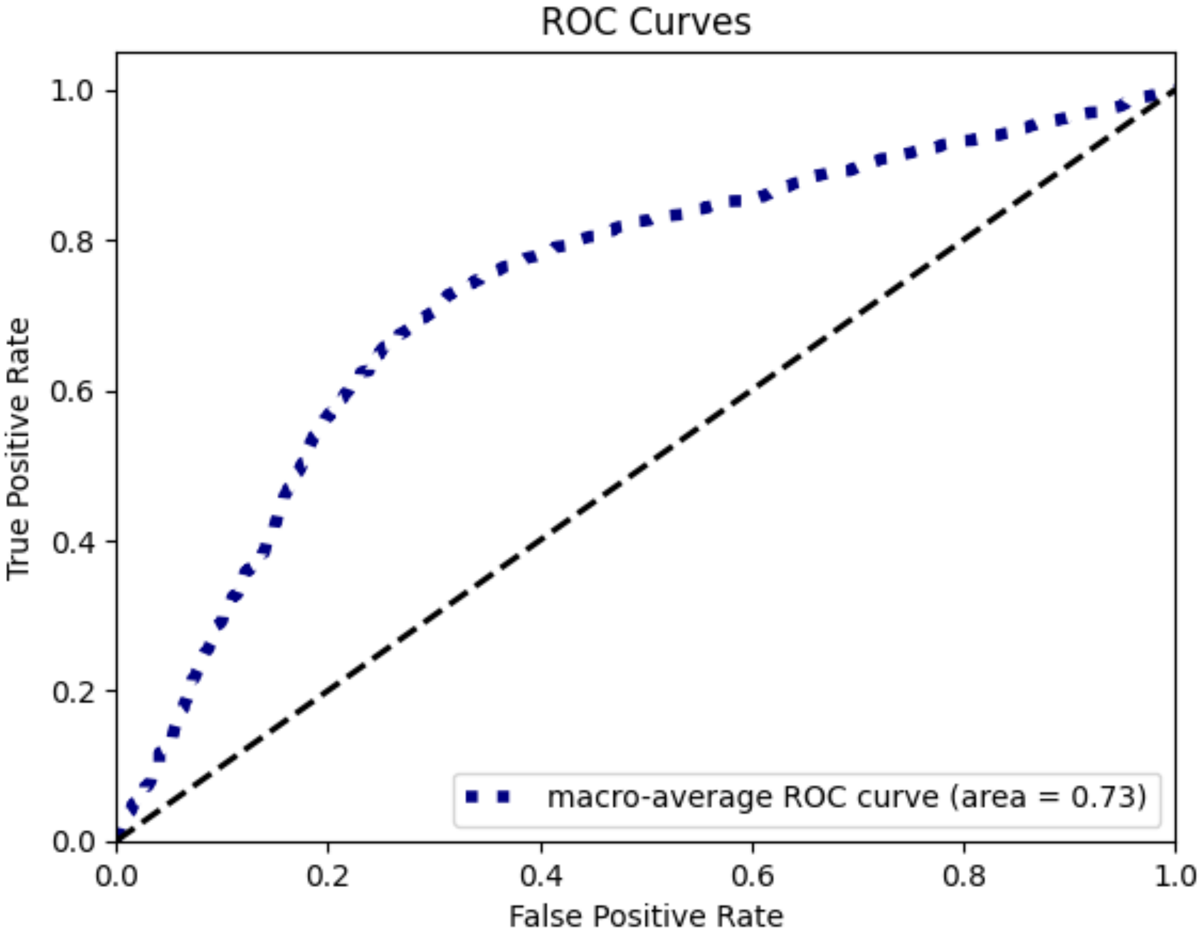


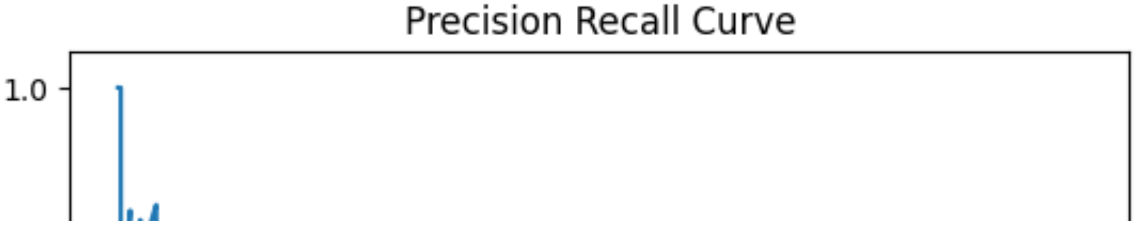


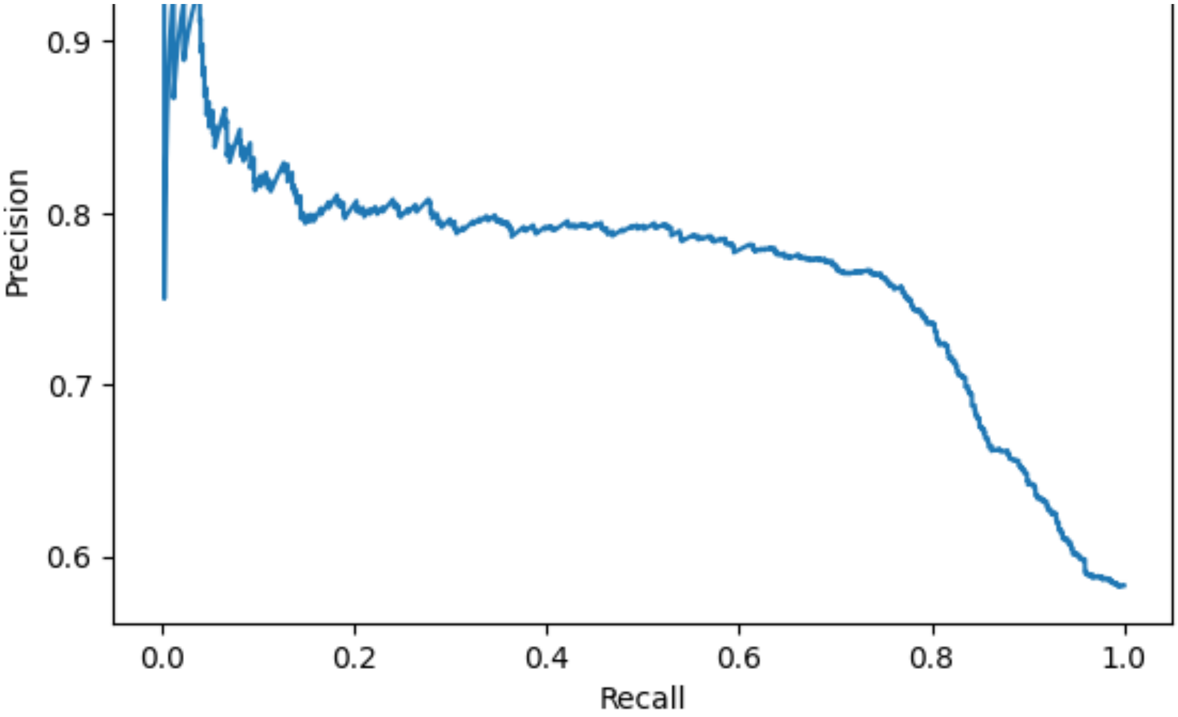


```
# 5 minute model
acc_5min_art, roc_auc_5min_art, pr_auc_5min_art, y_true_5min_art, y_score_comb_5m
print_eval_parameters(test_loader_5min, art_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_5min_art.pth")
```

```
Test Accuracy: 0.4168552036199095
Test AUC: 0.7328778030313997
Test PR AUC: 0.7679671901448024
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

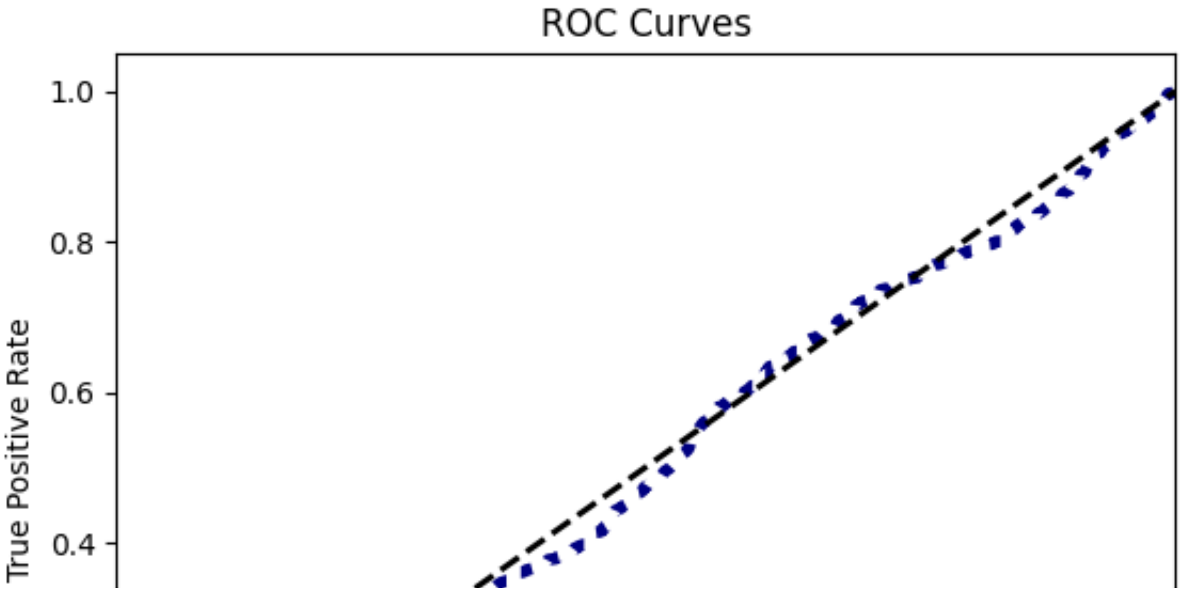


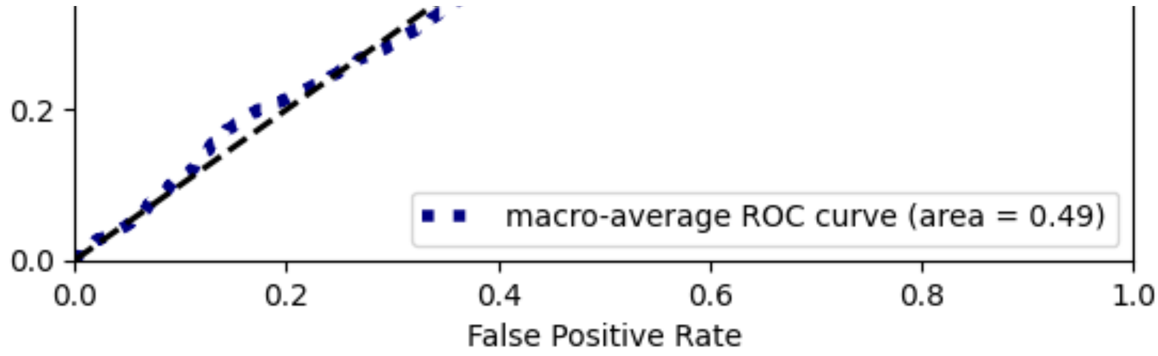


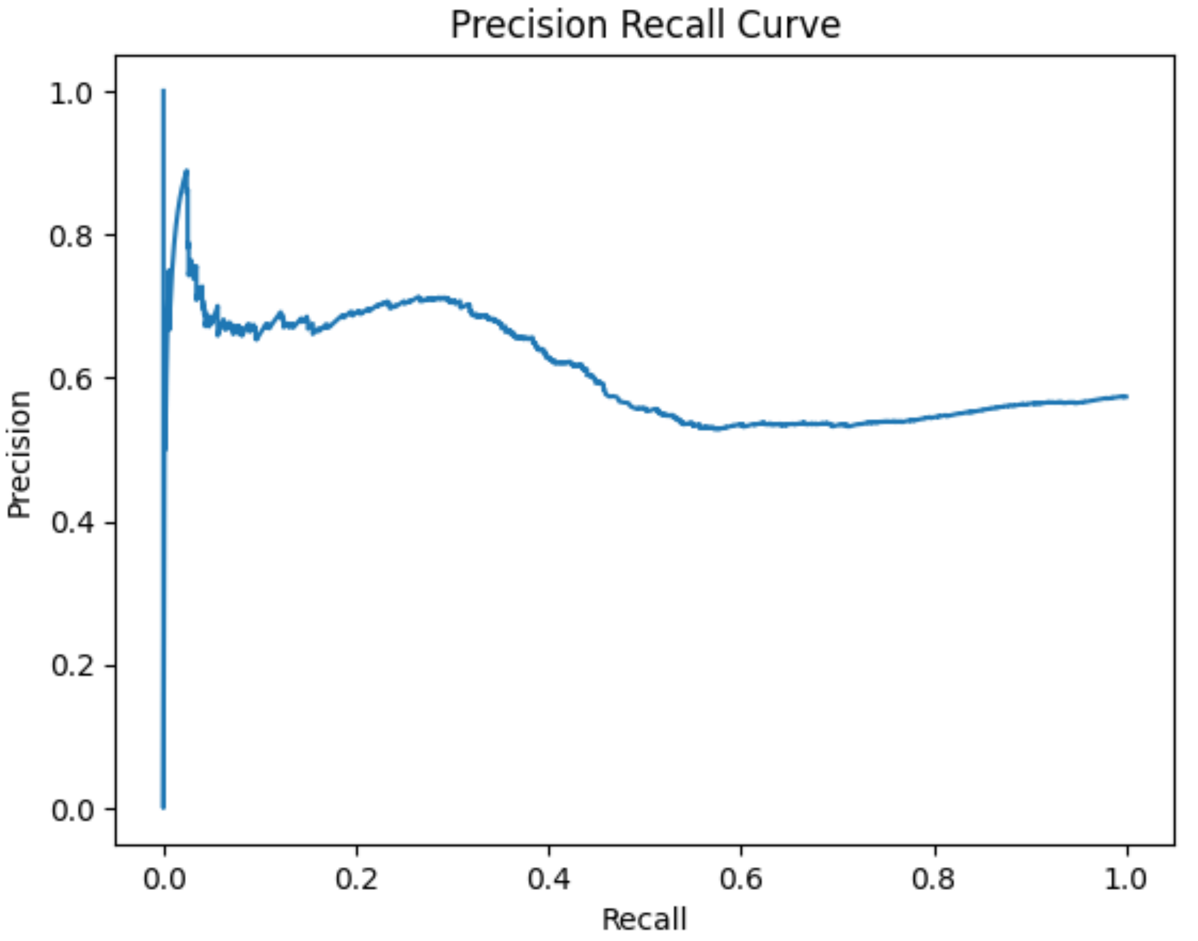


```
# 10 minute model
acc_10min_art, roc_auc_10min_art, pr_auc_10min_art, y_true_10min_art, y_score_com
print_eval_parameters(test_loader_10min, art_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_10min_art.pth")
```

```
Test Accuracy: 0.4266743648960739
Test AUC: 0.49343850798621475
Test PR AUC: 0.6094249133090233
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```





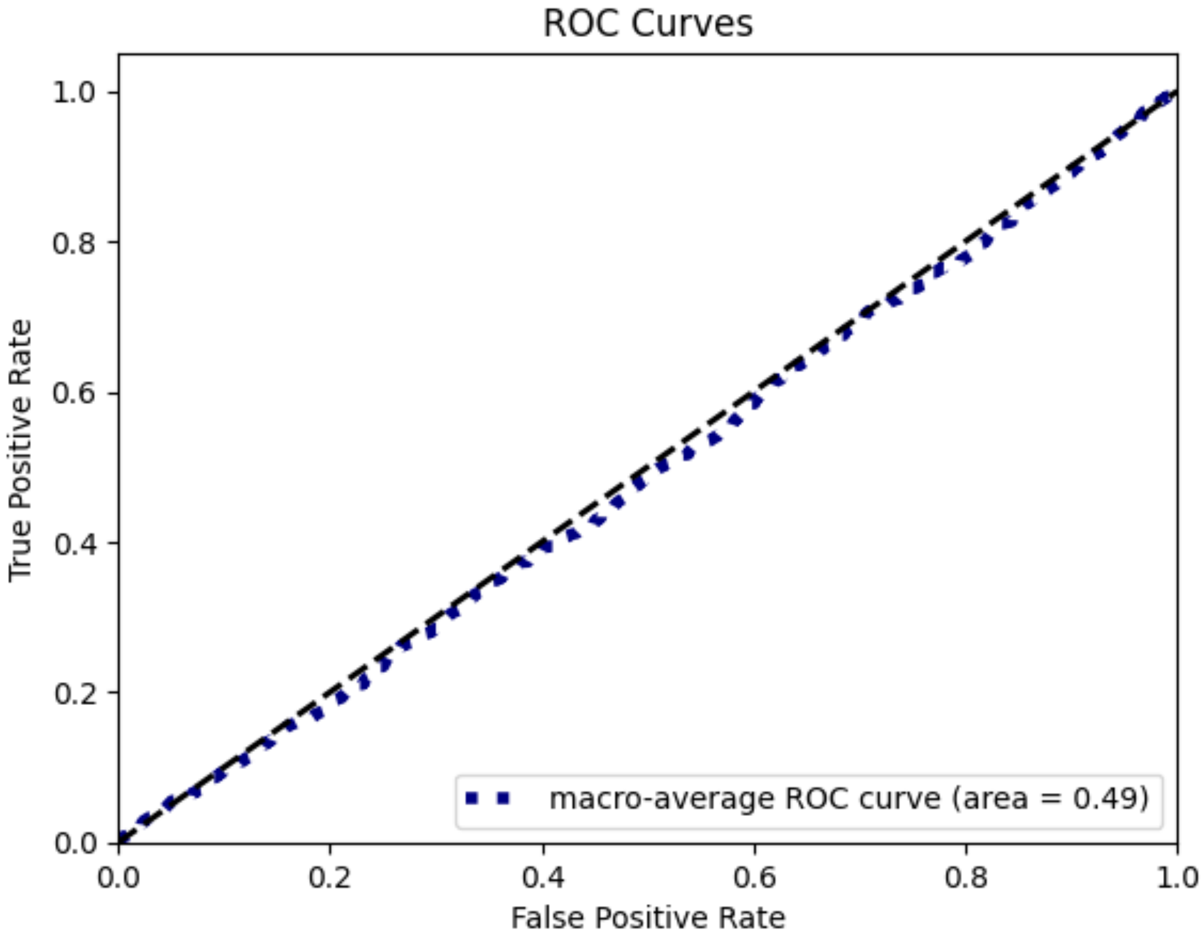


```
# 15 minute model
acc_15min_art, roc_auc_15min_art, pr_auc_15min_art, y_true_15min_art, y_score_com
print_eval_parameters(test_loader_15min, art_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_15min_art.pth")
```

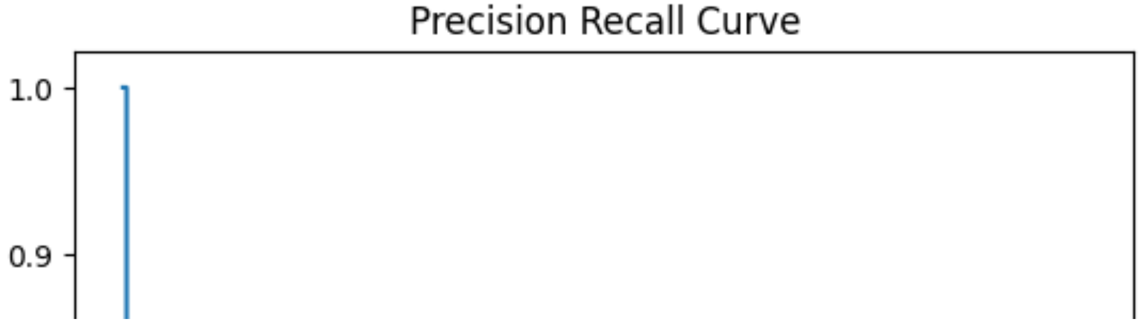
### EEG model

```
# 3 minute model
acc_3min_eeg, roc_auc_3min_eeg, pr_auc_3min_eeg, y_true_3min_eeg, y_score_comb_3m
print_eval_parameters(test_loader_3min, eeg_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_3min_eeg.pth")
```

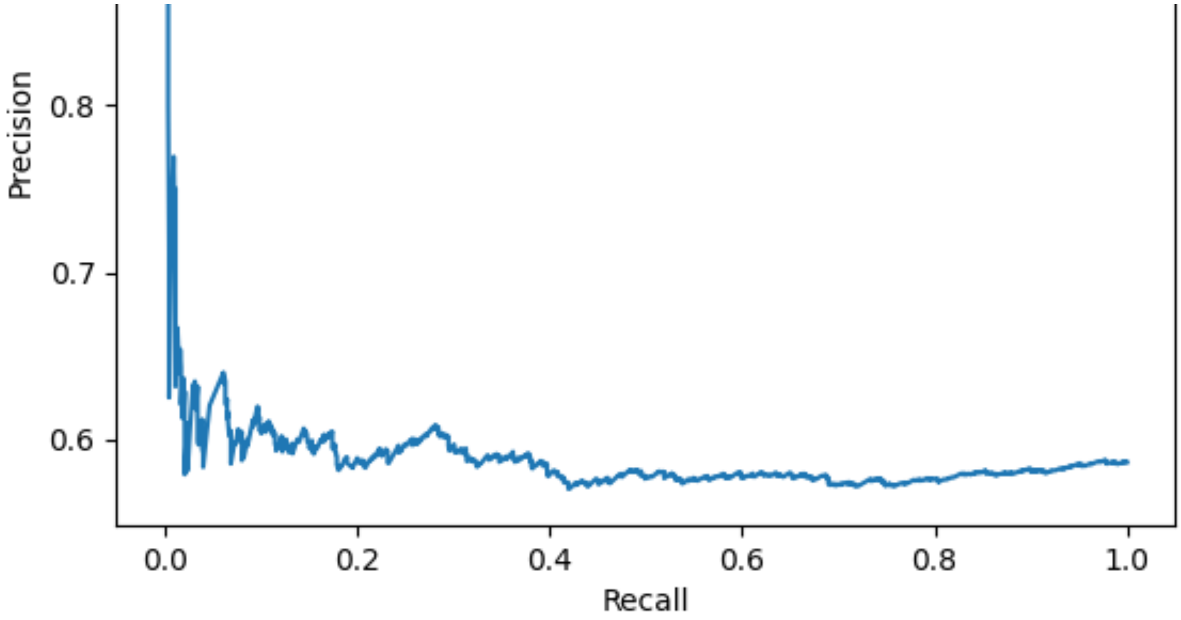
```
Test Accuracy: 0.5856260527793374
Test AUC: 0.4890171659986379
Test PR AUC: 0.5892474409496379
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```



ROC Curves

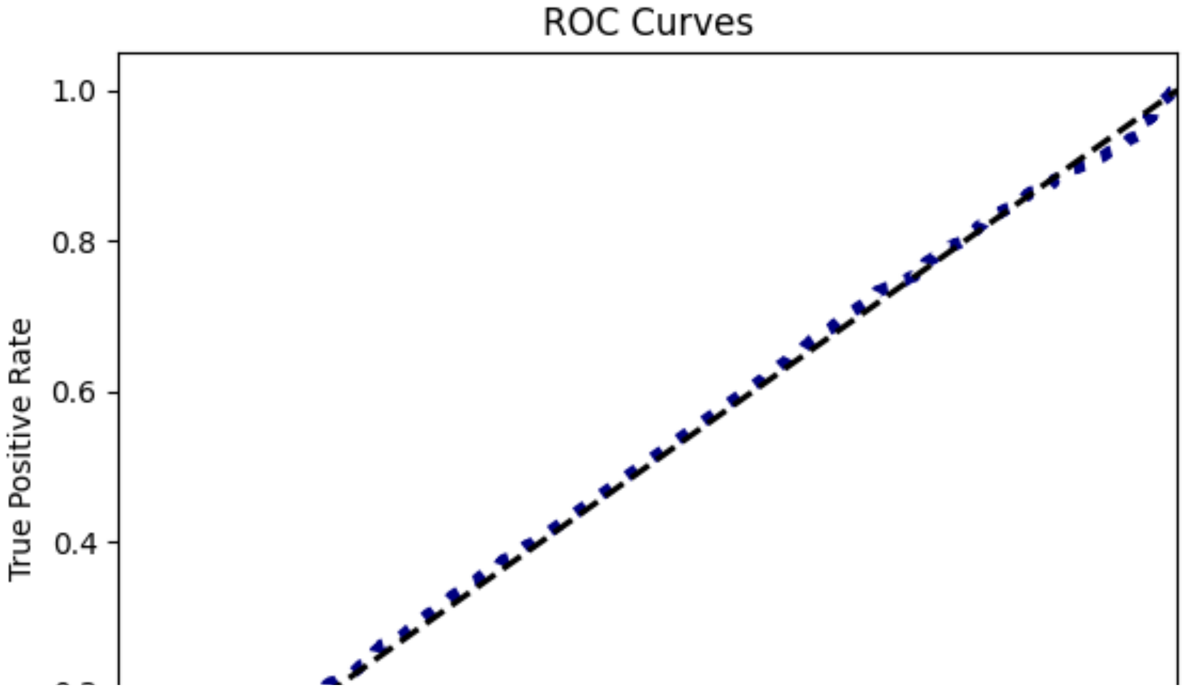


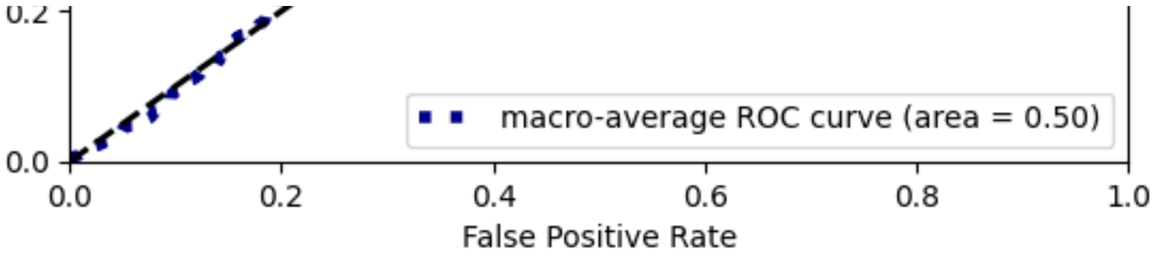
Precision Recall Curve

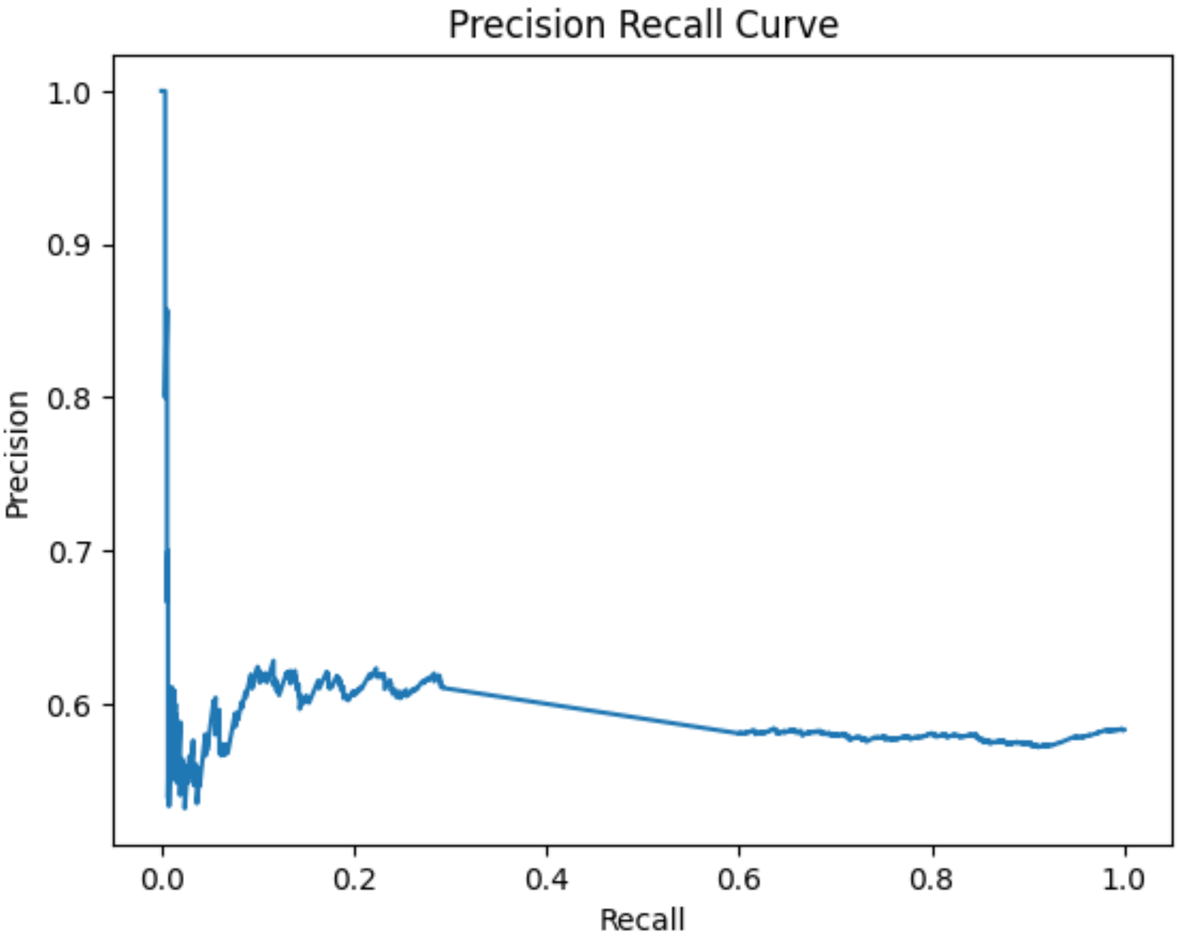


```
# 5 minute model
acc_5min_eeg, roc_auc_5min_eeg, pr_auc_5min_eeg, y_true_5min_eeg, y_score_comb_5m
print_eval_parameters(test_loader_5min, eeg_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_5min_eeg.pth")
```

```
Test Accuracy: 0.5831447963800905
Test AUC: 0.5022287381538653
Test PR AUC: 0.5887226929898296
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

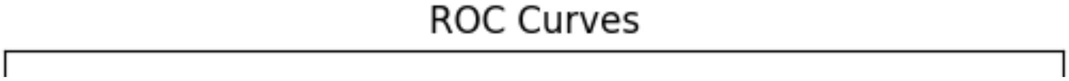


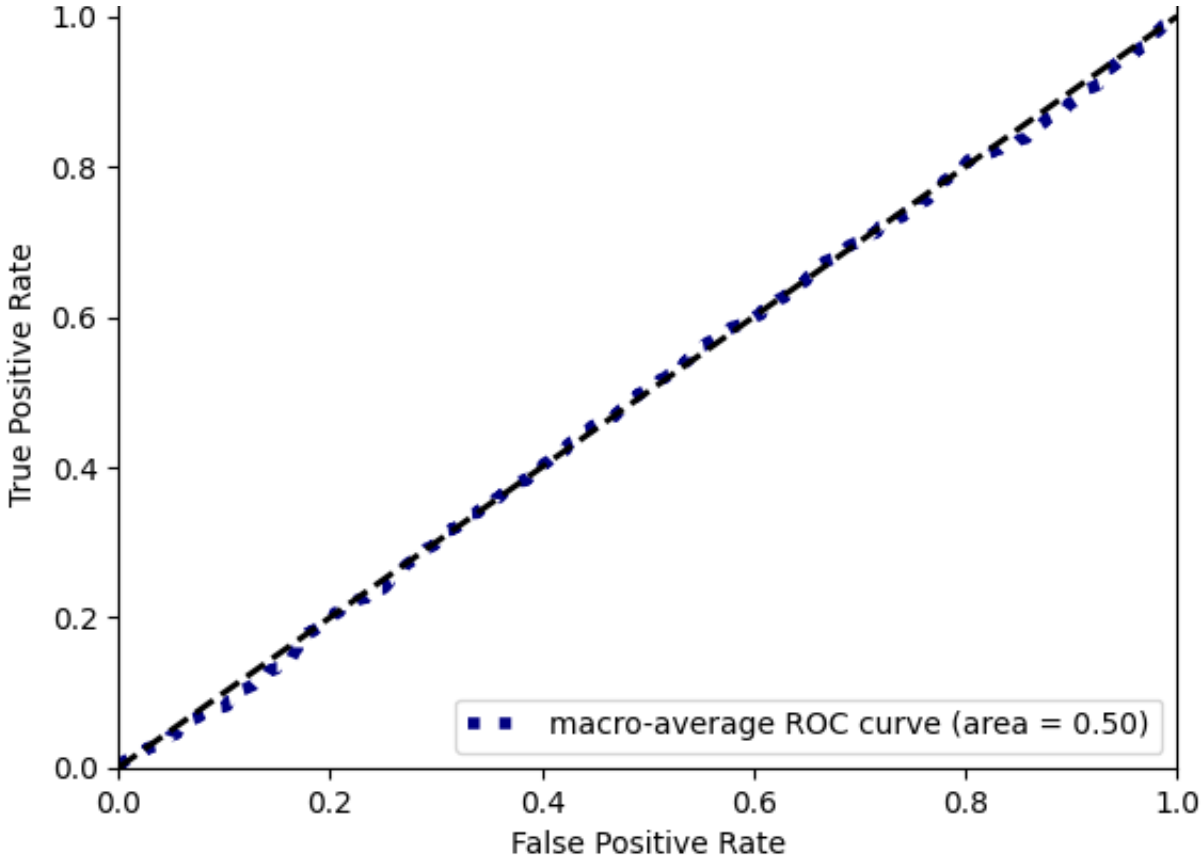


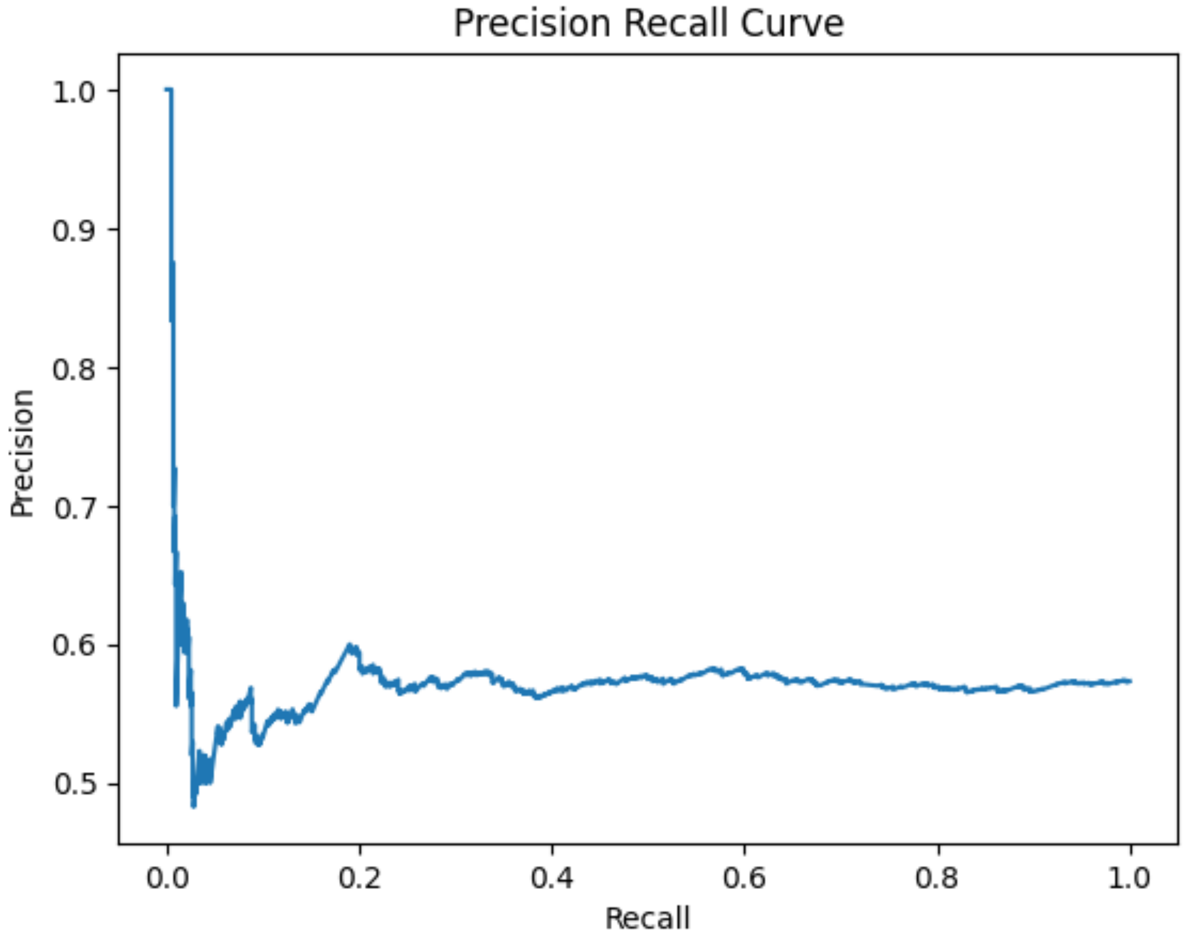


```
# 10 minute model
acc_10min_eeg, roc_auc_10min_eeg, pr_auc_10min_eeg, y_true_10min_eeg, y_score_com
print_eval_parameters(test_loader_10min, eeg_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_10min_eeg.pth")
```

```
Test Accuracy: 0.5733256351039261
Test AUC: 0.496053565758687
Test PR AUC: 0.5726342266881312
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

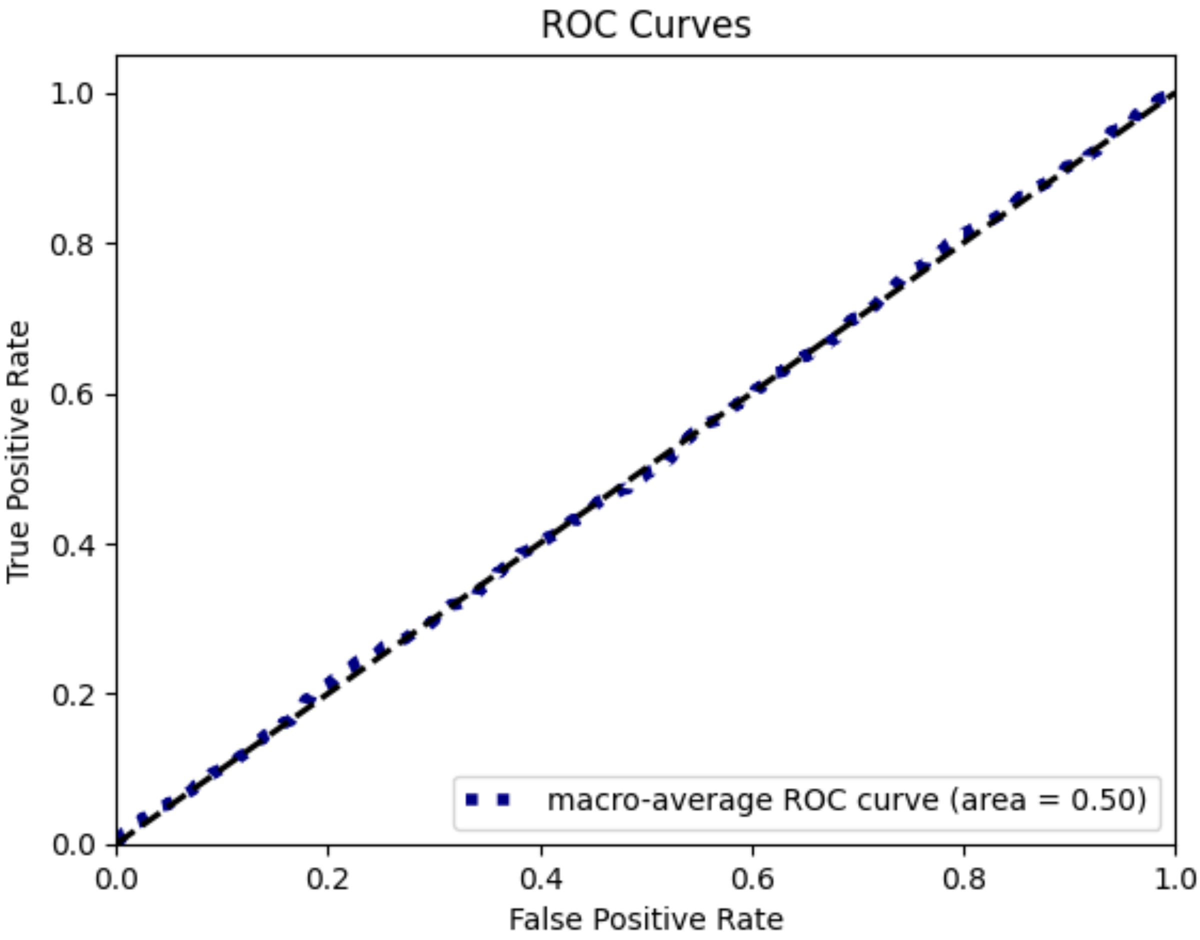


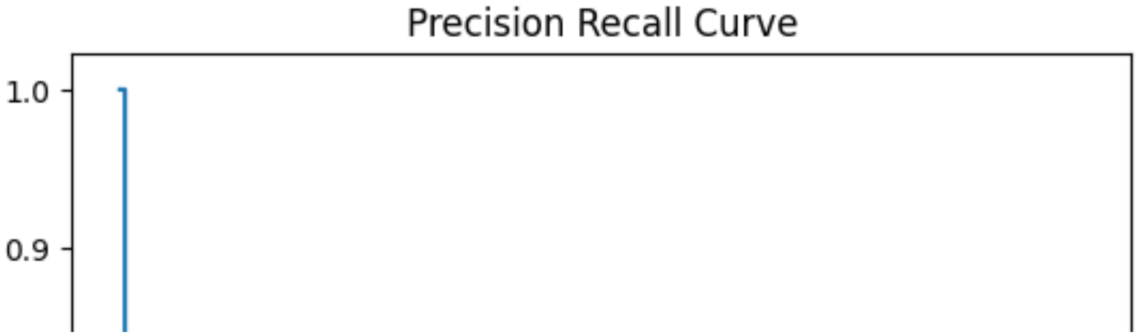

True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
macro-average ROC curve (area = 0.50)

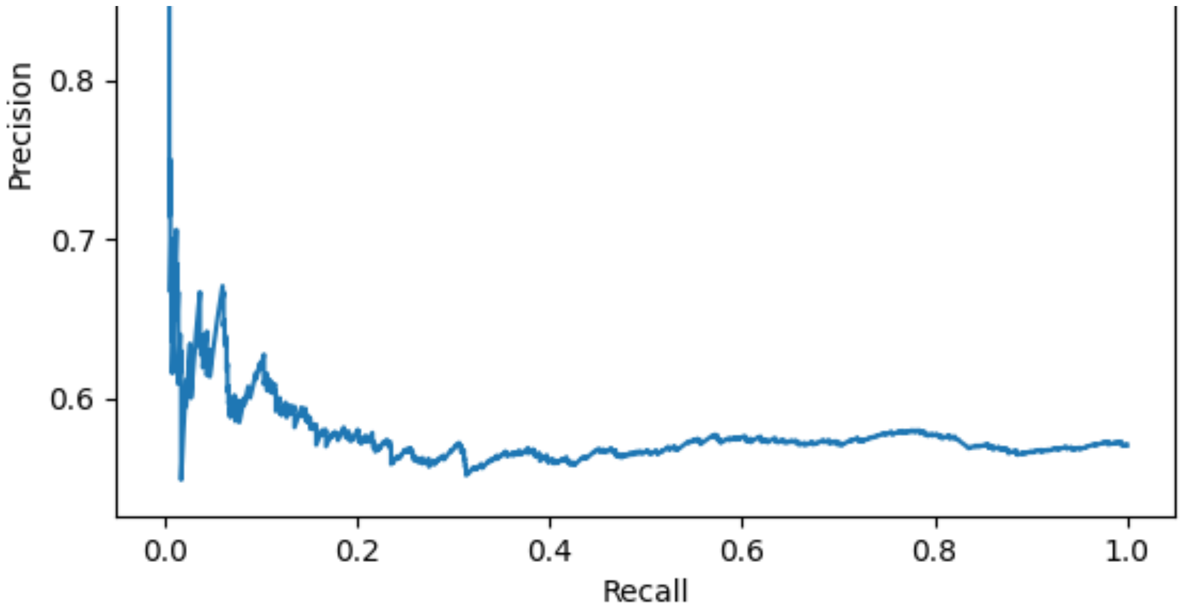

Precision Recall Curve
Precision
Recall
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.2
0.4
0.6
0.8
1.0

```
# 15 minute model
acc_15min_eeg, roc_auc_15min_eeg, pr_auc_15min_eeg, y_true_15min_eeg, y_score_com
print_eval_parameters(test_loader_15min, eeg_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_15min_eeg.pth")
```

```
Test Accuracy: 0.5688448074679113
Test AUC: 0.5013978215371454
Test PR AUC: 0.5786422871041599
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```



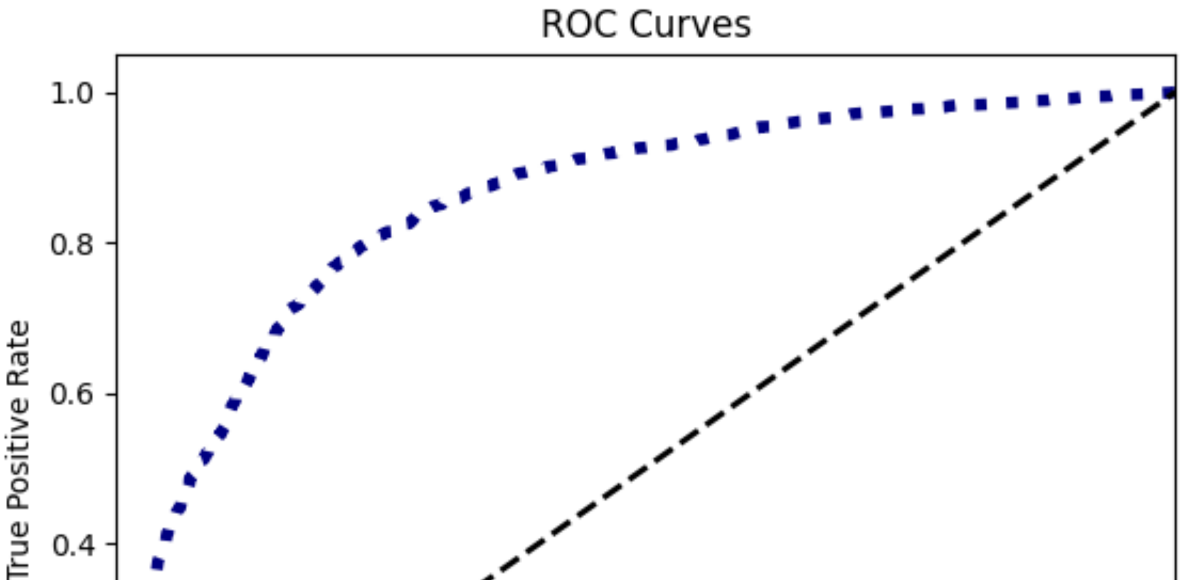


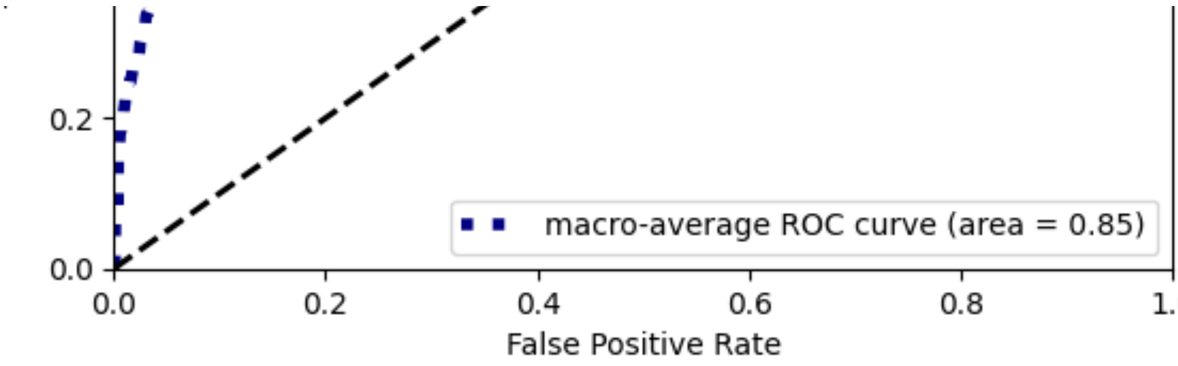


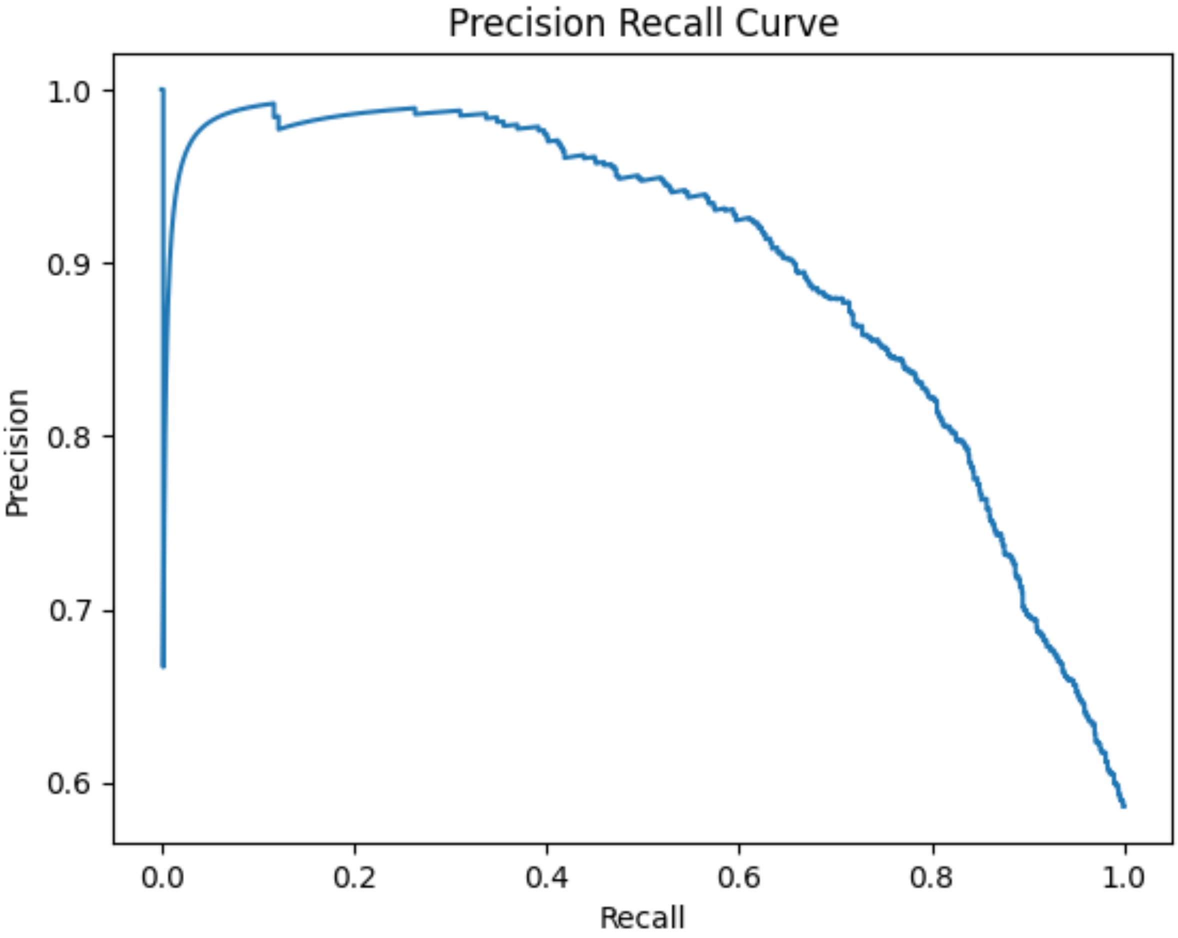
### Shallow model

```
# 3 minute model
acc_3min_shallow, roc_auc_3min_shallow, pr_auc_3min_shallow, y_true_3min_shallow,
print_eval_parameters(test_loader_3min, shallow_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_3min_shallow.pth
```

```
Test Accuracy: 0.7815833801235261
Test AUC: 0.8505773119772091
Test PR AUC: 0.8987637458560864
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

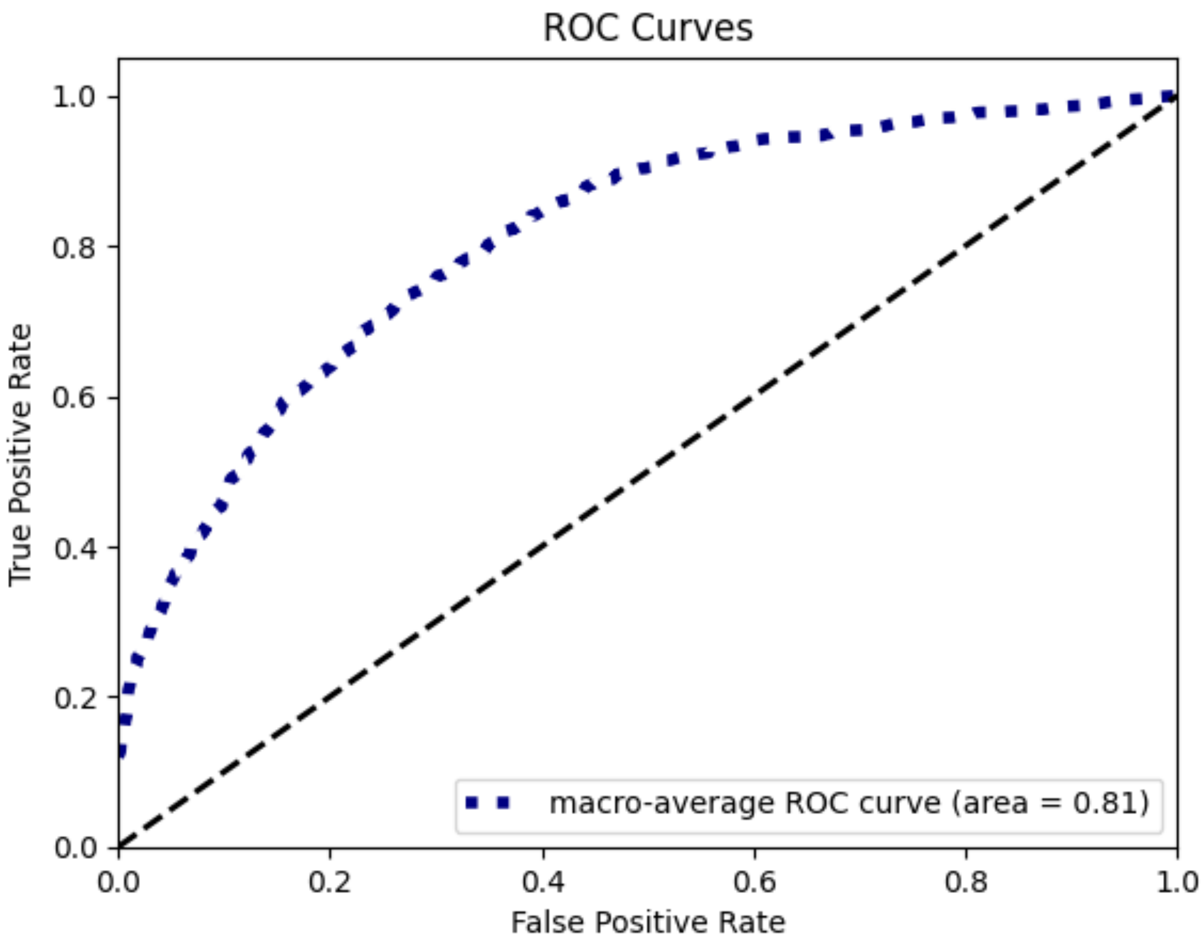


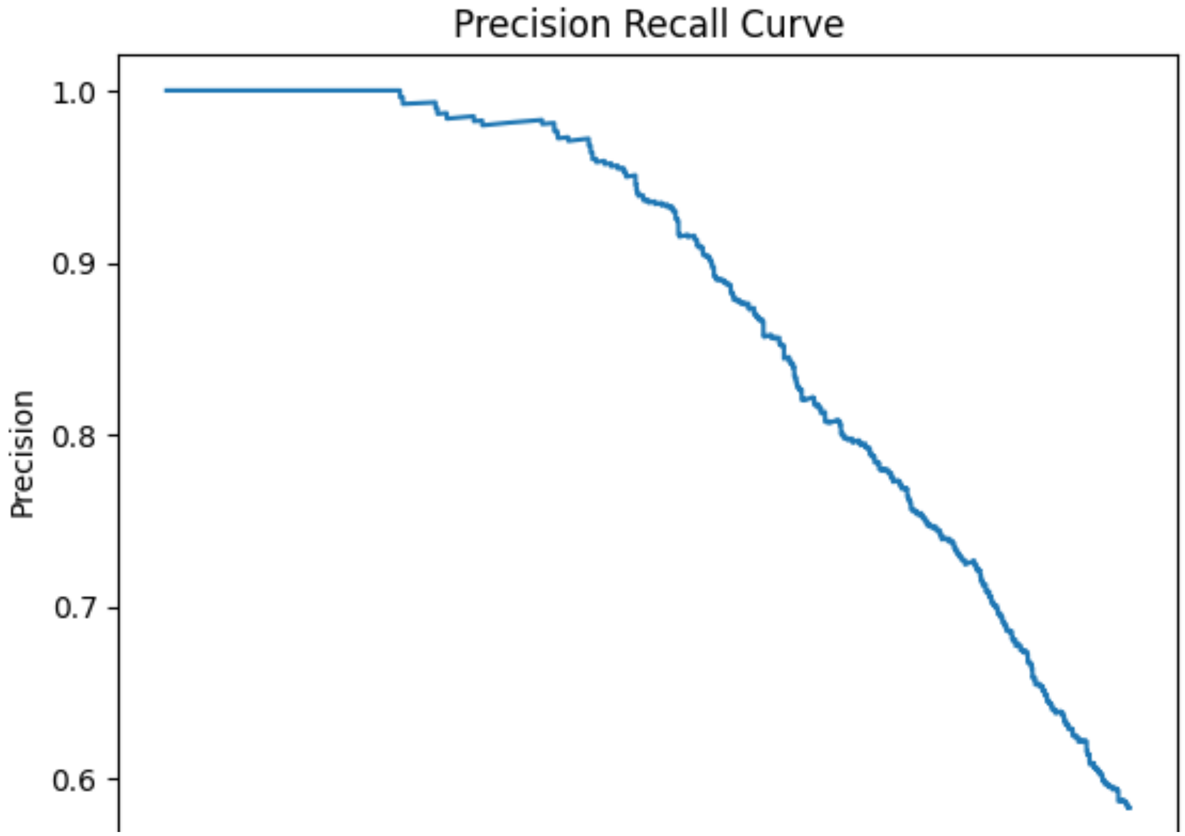


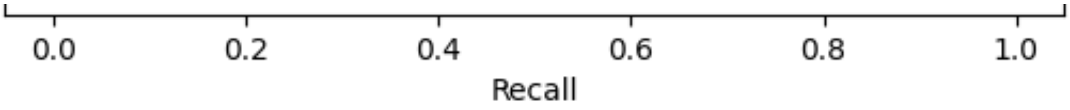


```
# 5 minute model
acc_5min_shallow, roc_auc_5min_shallow, pr_auc_5min_shallow, y_true_5min_shallow,
print_eval_parameters(test_loader_5min, shallow_model(), 0.001, torch.optim.Adam,
                      "/content/drive/MyDrive/VitalDB/best_model_5min_shallow.pth
```

```
Test Accuracy: 0.7273755656108597
Test AUC: 0.808727283255708
Test PR AUC: 0.8788752368624182
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
warnings.warn(msg, category=FutureWarning)
```

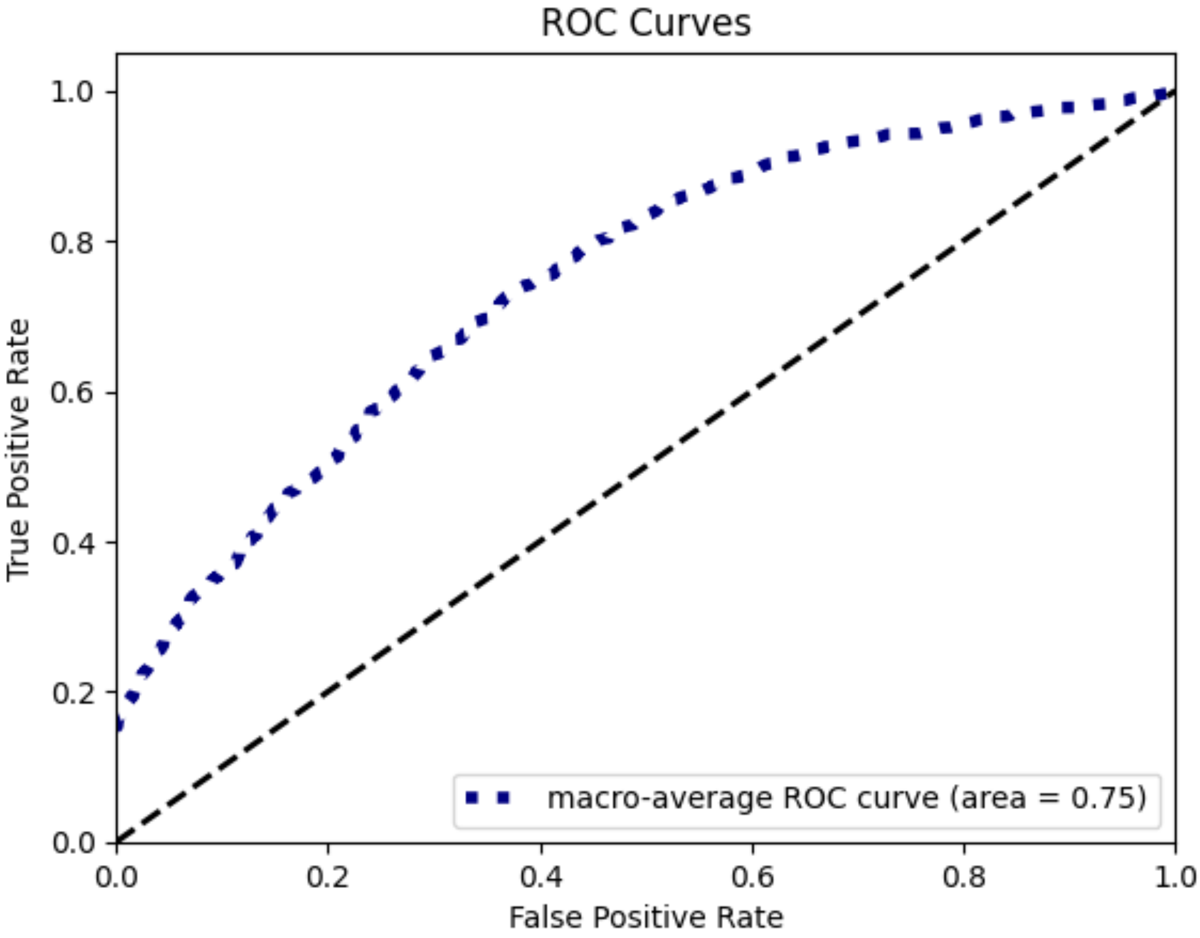




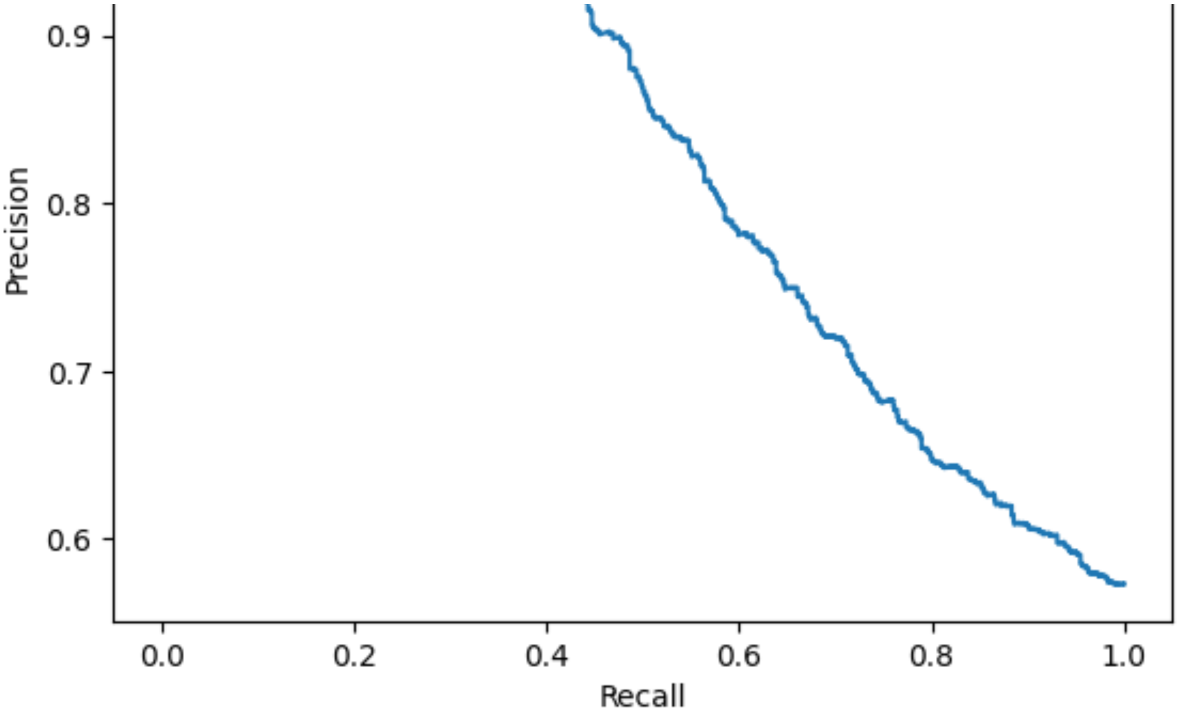


```
# 10 minute model
acc_10min_shallow, roc_auc_10min_shallow, pr_auc_10min_shallow, y_true_10min_shal
print_eval_parameters(test_loader_10min, shallow_model(), 0.001, torch.optim.Adam
                      "/content/drive/MyDrive/VitalDB/best_model_10min_shallow.pt
```

```
Test Accuracy: 0.6766743648960739
Test AUC: 0.745335753522288
Test PR AUC: 0.8376363480152185
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

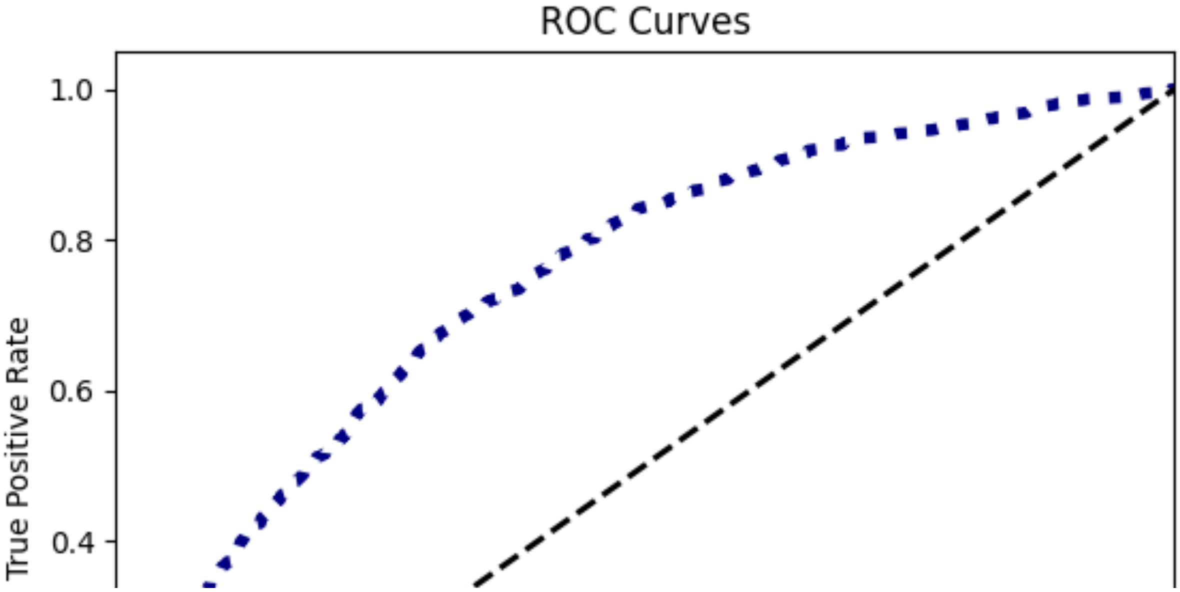


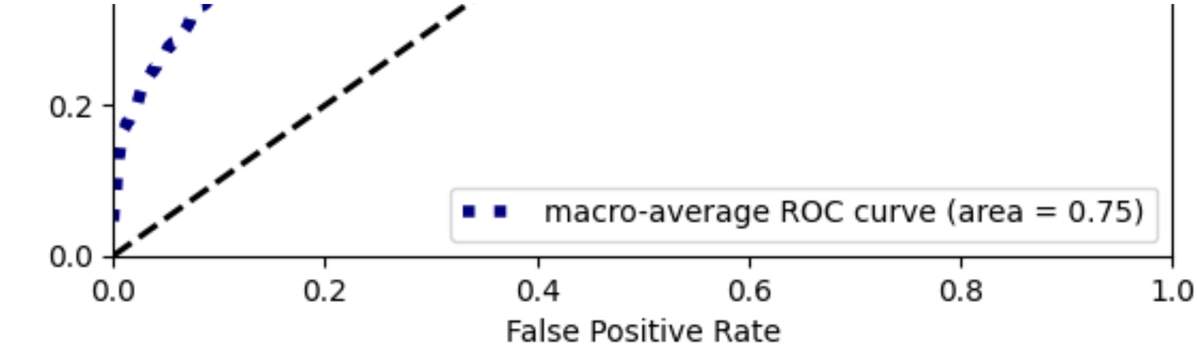
Precision Recall Curve
1.0

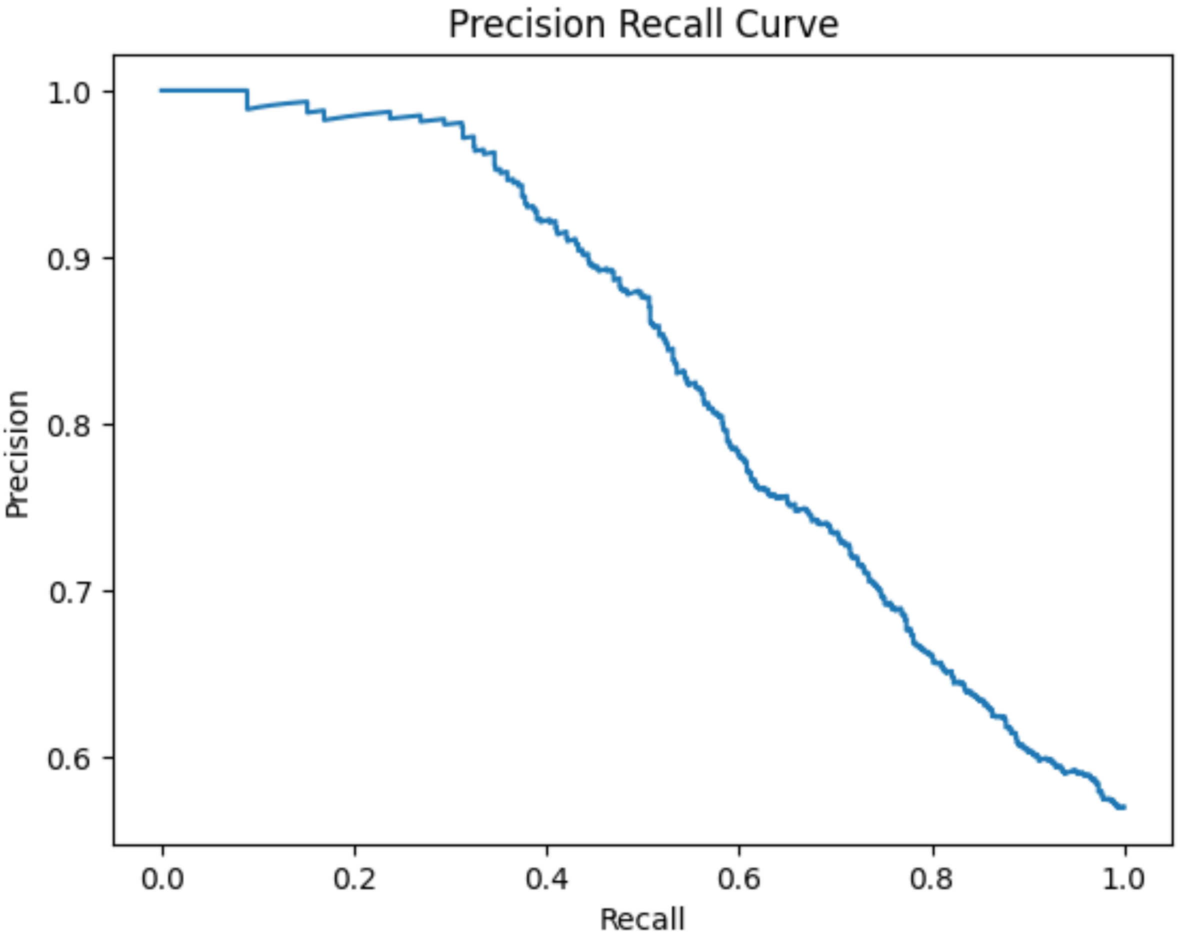


```
# 15 minute model
acc_15min_shallow, roc_auc_15min_shallow, pr_auc_15min_shallow, y_true_15min_shal
print_eval_parameters(test_loader_15min, shallow_model(), 0.001, torch.optim.Adam
                      "/content/drive/MyDrive/VitalDB/best_model_15min_shallow.pt
```

```
Test Accuracy: 0.676779463243874
Test AUC: 0.7489976376607703
Test PR AUC: 0.8322632438848699
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```



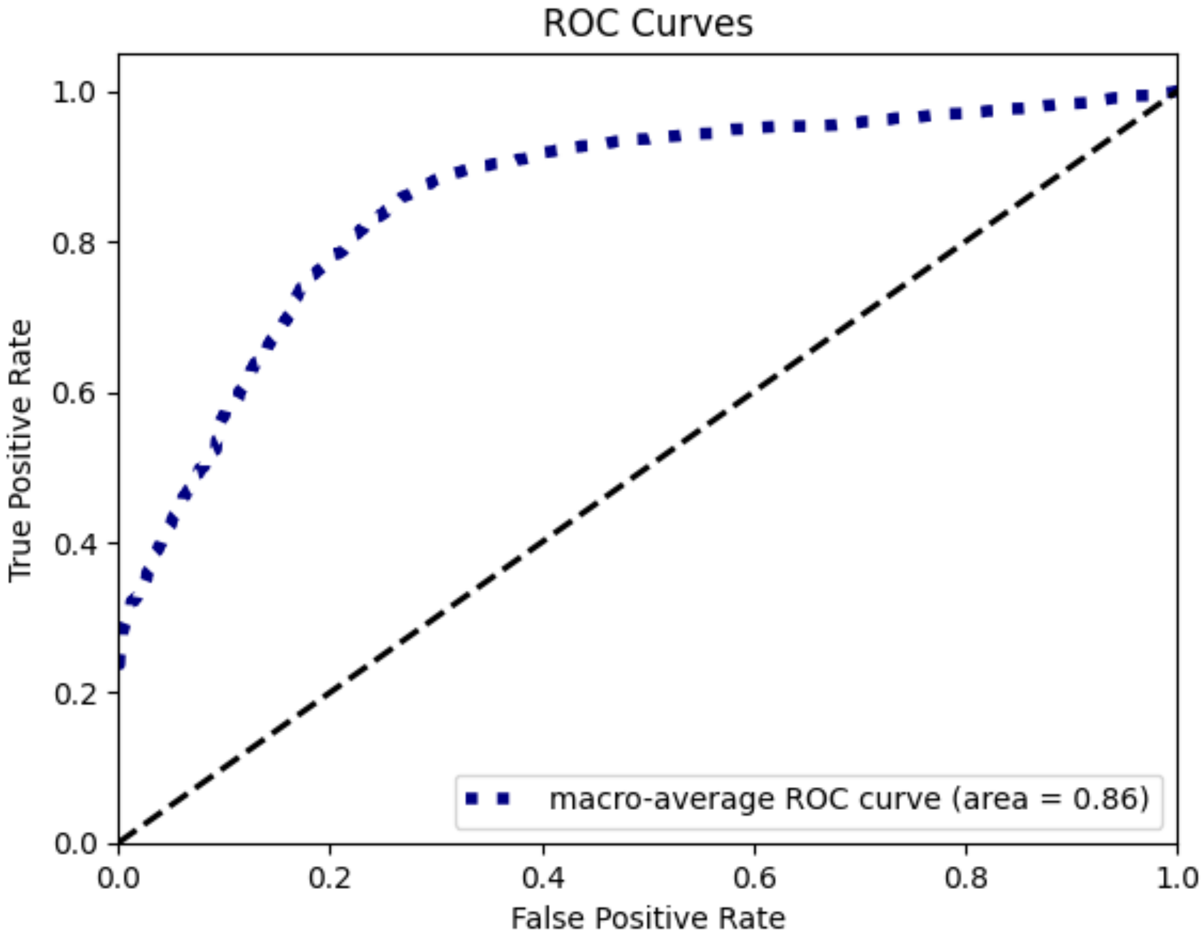


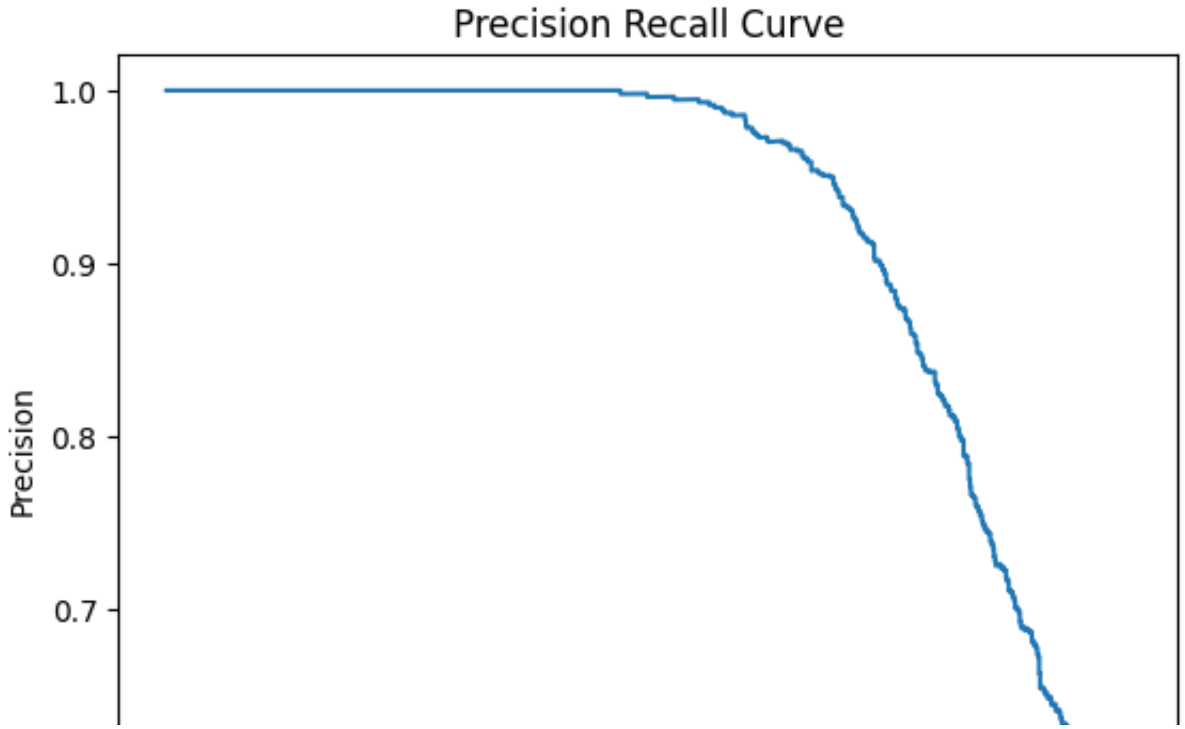


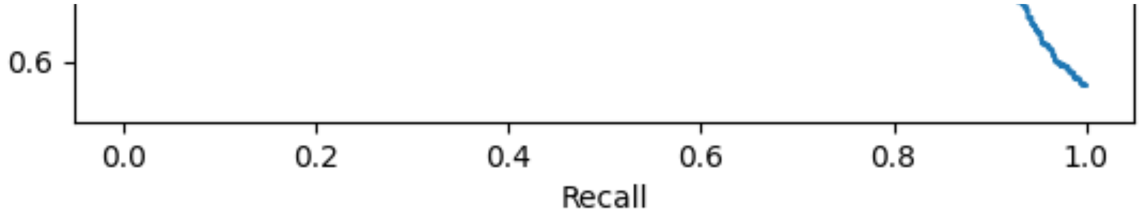
### Model with leaky Relu activation

```
# 3 minute model
acc_3min_leaky_relu, roc_auc_3min_leaky_relu, pr_auc_3min_leaky_relu, y_true_3mir
print_eval_parameters(test_loader_3min, model_leaky_relu(), 0.001, torch.optim.Ac
                      "/content/drive/MyDrive/VitalDB/best_model_3min_leaky_relu.
```

```
Test Accuracy: 0.7995508141493542
Test AUC: 0.8589094756104534
Test PR AUC: 0.9217427597214178
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

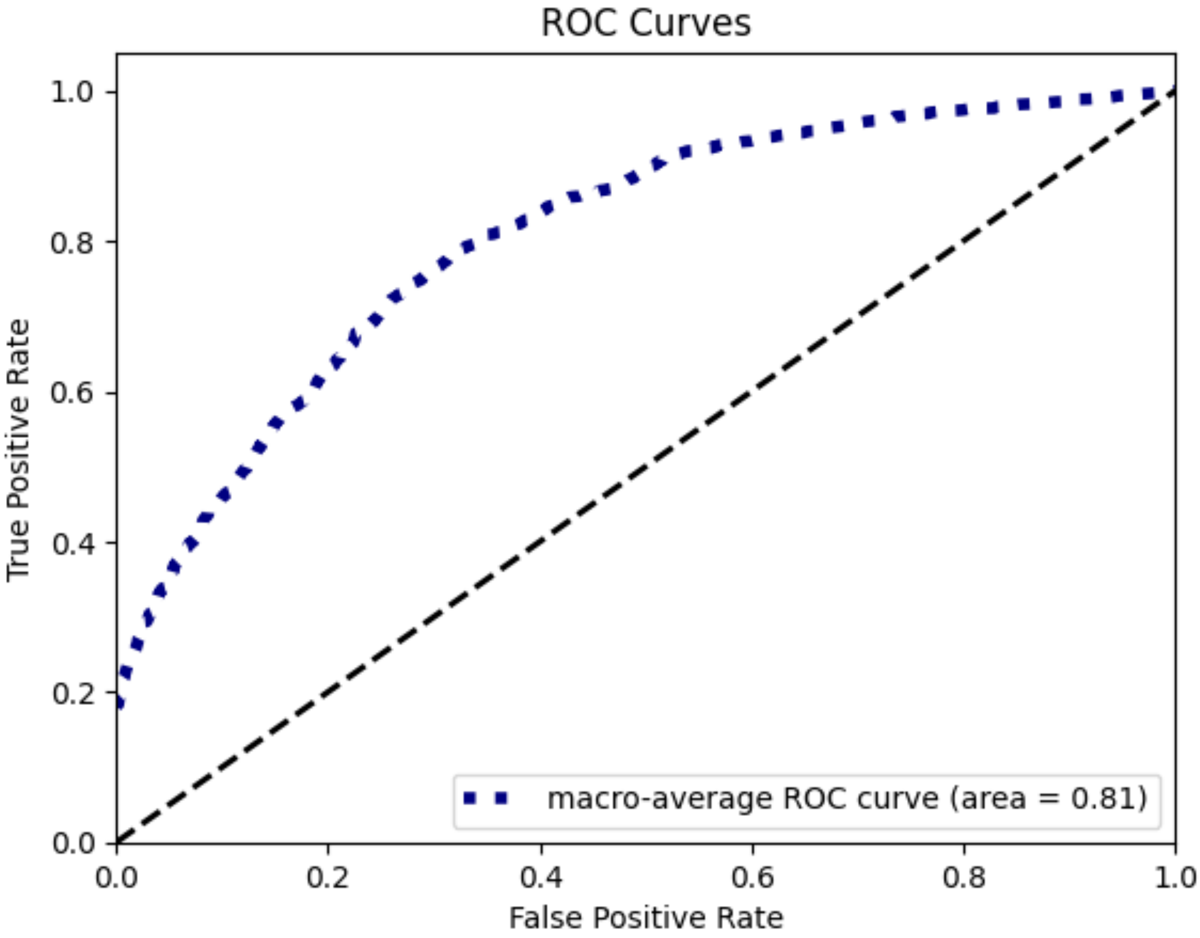


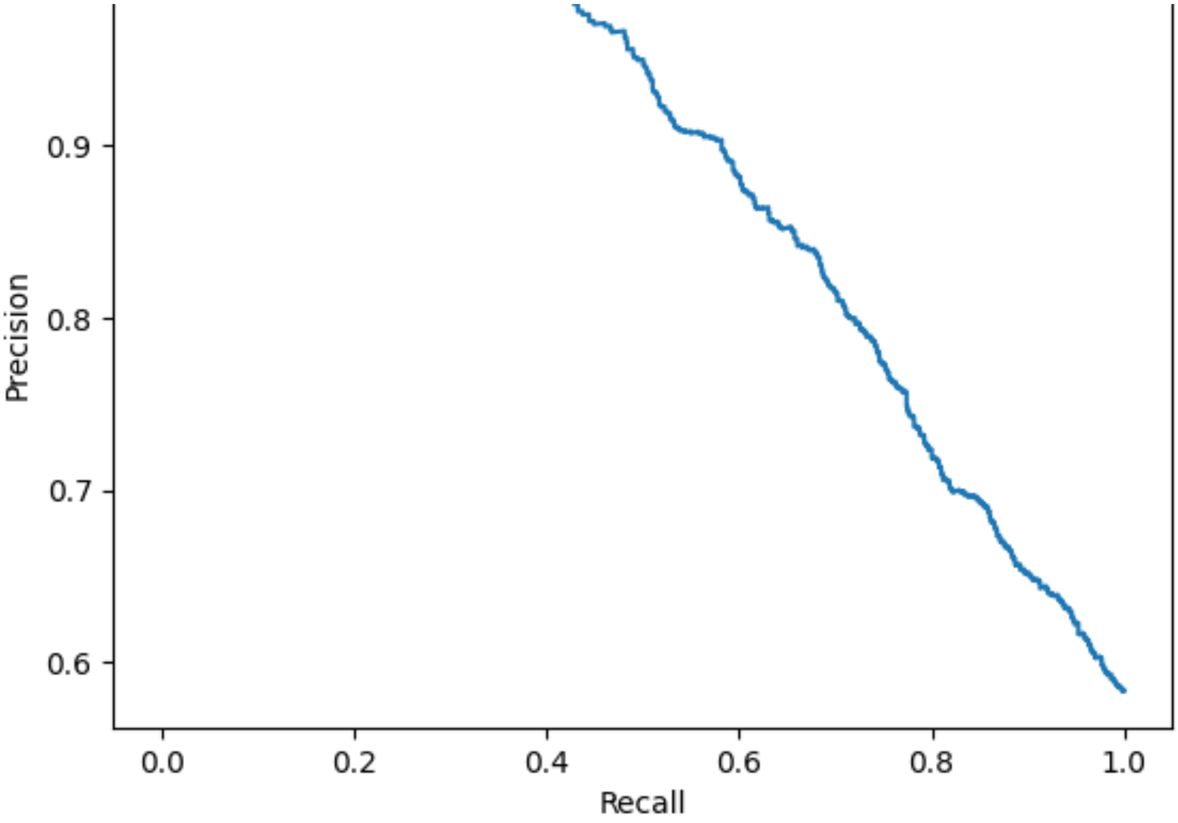




```
# 5 minute model
acc_5min_leaky_relu, roc_auc_5min_leaky_relu, pr_auc_5min_leaky_relu, y_true_5mir
print_eval_parameters(test_loader_5min, model_leaky_relu(), 0.001, torch.optim.Ac
                      "/content/drive/MyDrive/VitalDB/best_model_5min_leaky_relu.
```

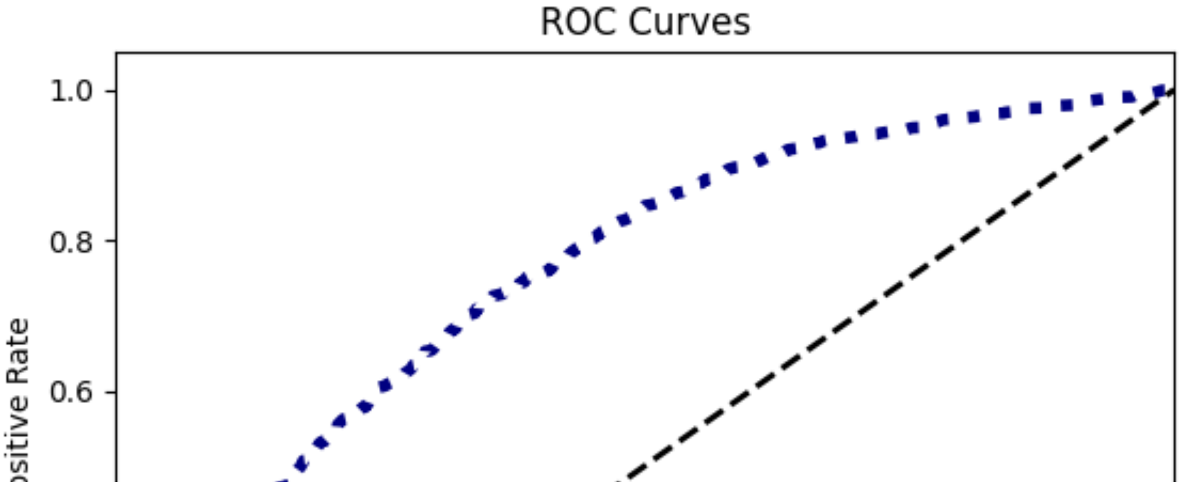
```
Test Accuracy: 0.6945701357466063
Test AUC: 0.806342592653521
Test PR AUC: 0.8809655791589137
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

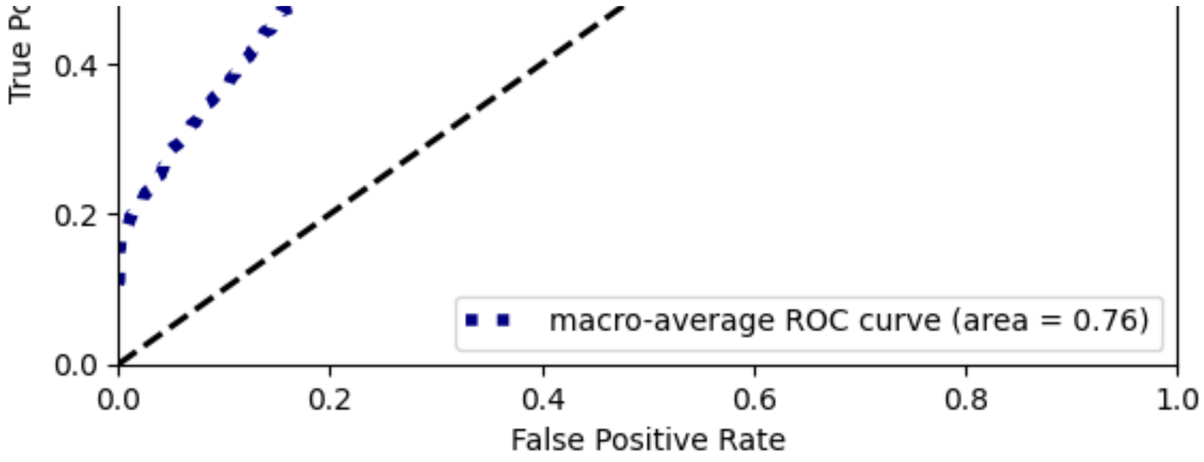


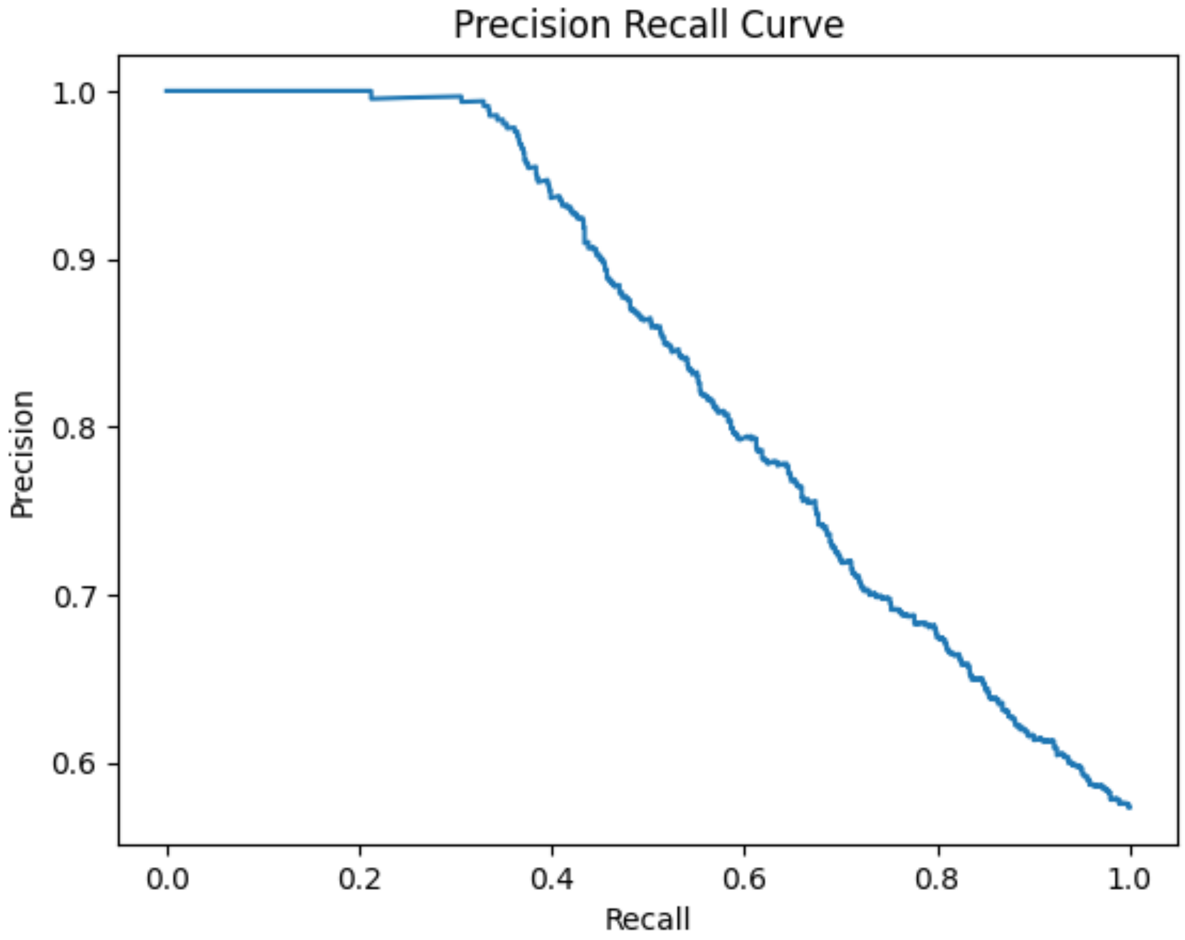


```
# 10 minute model
acc_10min_leaky_relu, roc_auc_10min_leaky_relu, pr_auc_10min_leaky_relu, y_true_1
print_eval_parameters(test_loader_10min, model_leaky_relu(), 0.001, torch.optim.A
                      "/content/drive/MyDrive/VitalDB/best_model_10min_leaky_relu
```

```
Test Accuracy: 0.6564665127020786
Test AUC: 0.7546547074446702
Test PR AUC: 0.8402394851714581
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```

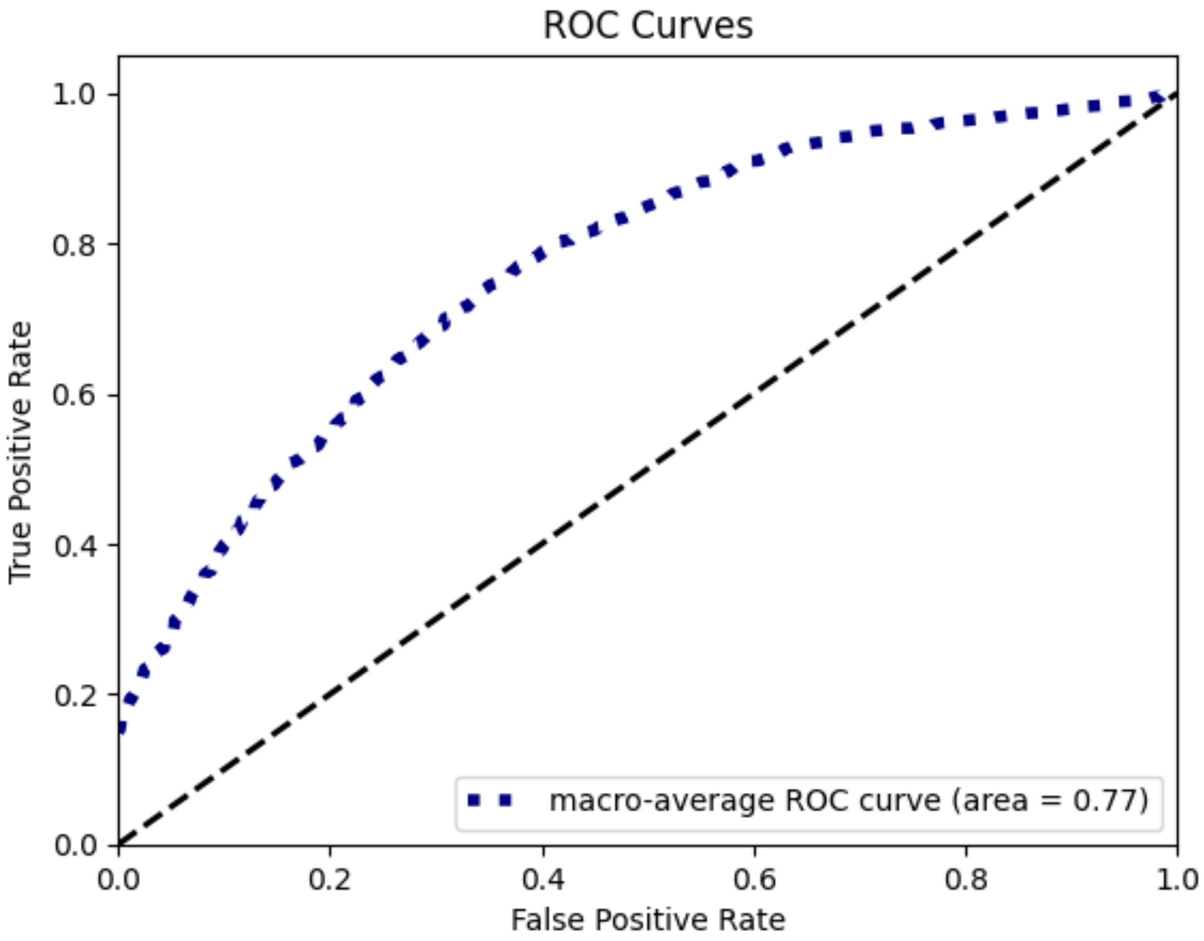


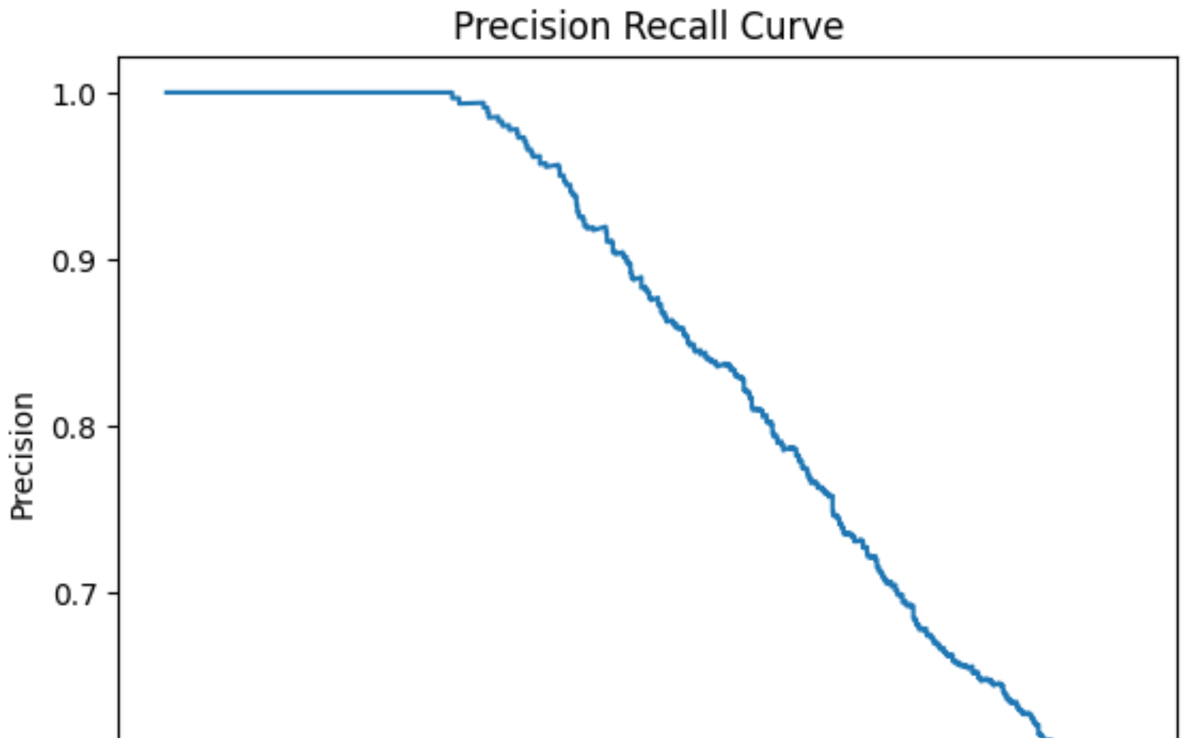




```
# 15 minute model
acc_15min_leaky_relu, roc_auc_15min_leaky_relu, pr_auc_15min_leaky_relu, y_true_1
print_eval_parameters(test_loader_15min, model_leaky_relu(), 0.001, torch.optim.A
                      "/content/drive/MyDrive/VitalDB/best_model_15min_leaky_relu
```

```
Test Accuracy: 0.6674445740956826
Test AUC: 0.7663020155572746
Test PR AUC: 0.8477526311606528
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: Futu
  warnings.warn(msg, category=FutureWarning)
```





## Model comparison

```
# compare you model with others
# you don't need to re-run all other experiments, instead, you can directly refer
```

*1) Compare main model to model originally published*

The results of the main analysis (model combining ECG, ART, and EEG) are slightly worse than the performance of the model in the original paper:

3 min:

- Original AUROC: 0.957, original AUPRC: 0.926
- Reproduced AUROC: 0.852, reproduced AUPRC: 0.915

  5 min:

- Original AUROC: 0.926, original AUPRC: 0.867
- Reproduced AUROC: 0.800, reproduced AUPRC: 0.878

  10 min:

- Original AUROC: 0.895, original AUPRC: 0.817
- Reproduced AUROC: 0.756, reproduced AUPRC: 0.840

  15 min:

- Original AUROC: 0.868, original AUPRC: 0.778
- Reproduced AUROC: 0.761, reproduced AUPRC: 0.843

Accuracy was not reported in the original paper

*2) Compare ECG only models*

3 min:

- Original AUROC: 0.634, original AUPRC: 0.339
- Reproduced AUROC: 0.483, reproduced AUPRC: 0.579

  5 min:

- Original AUROC: 0.652, original AUPRC: 0.359
- Reproduced AUROC: 0.521, reproduced AUPRC: 0.606

  10 min:

- Original AUROC: 0.659, original AUPRC: 0.364
- Reproduced AUROC: 0.472, reproduced AUPRC: 0.553

  15 min:

- Original AUROC: 0.640, original AUPRC: 0.306
- Reproduced AUROC: 0.495, reproduced AUPRC: 0.572

*3) Compare ART only models*

3 min:

- Original AUROC: 0.968, original AUPRC: 0.939
- Reproduced AUROC: 0.823, reproduced AUPRC: 0.876

  5 min:

- Original AUROC: 0.930, original AUPRC: 0.873
- Reproduced AUROC: 0.733, reproduced AUPRC: 0.770

  10 min:

- Original AUROC: 0.892, original AUPRC: 0.814
- Reproduced AUROC: 0.493, reproduced AUPRC: 0.609

  15 min:

- Original AUROC: 0.889, original AUPRC: 0.803
- Reproduced AUROC: 0.501, reproduced AUPRC: 0.579

*4) Compare EEG only models*

3 min:

- Original AUROC: 0.557, original AUPRC: 0.286
- Reproduced AUROC: 0.489, reproduced AUPRC: 0.589

  5 min:
- Original AUROC: 0.581, original AUPRC: 0.301
- Reproduced AUROC: 0.502, reproduced AUPRC: 0.589

  10 min:
- Original AUROC: 0.584, original AUPRC: 0.297
- Reproduced AUROC: 0.496, reproduced AUPRC: 0.573

  15 min:
- Reproduced AUROC: 0.577, reproduced AUPRC: 0.260
- Reproduced AUROC: 0.501, reproduced AUPRC: 0.579

*5) Shallow model (not evaluated in original paper):*

3 min:

- Reproduced AUROC: 0.850, reproduced AUPRC: 0.899

  5 min:
- Reproduced AUROC: 0.809, reproduced AUPRC: 0.879

  10 min:
- Reproduced AUROC: 0.745, reproduced AUPRC: 0.838

  15 min:
- Reproduced AUROC: 0.749, reproduced AUPRC: 0.832

*6) Model with leaky Relu activations (not evaluated in original paper):*

3 min:

- Reproduced AUROC: 0.859, reproduced AUPRC: 0.922

  5 min:
- Reproduced AUROC: 0.806, reproduced AUPRC: 0.881

10 min:

- Reproduced AUROC: 0.755, reproduced AUPRC: 0.840

15 min:

- Reproduced AUROC: 0.766, reproduced AUPRC: 0.848

# Discussion

The main takeaways from this analysis are:

- The reproduced metrics are somewhat worse than the ones originally published. Possible reasons include:
  - The larger dataset used in the original analysis (N=120,416 cases and controls across the training, validation, and test datasets vs 8,903 cases and controls used for the current analysis
  - Exclusion of patients with poor arterial blood pressure tracings (defined as a jSQI<0.8) in the original study. The jSQI was not evaluated here as the signal analysis code is written in Matlab, which cannot be easily incorporated in Colab. However, tracings with very low or very high blood pressures as defined in the jSQI algorithm were excluded from the current analysis
- Similar to findings in the original study, models trained based on 3 minute intervals prior to a hypotensive events consistently performed better than models trained based on tracings recorded 5, 10, or 15 min before a hypotensive event. This makes physiologic sense considering that it is much easier to predicts complications arising shortly after a measurment was obtained
- As opposed to the original paper, a learning rate of 0.001 seemed to lead to a better performance than a learning rate of 0.0001, possibly due to the different number of samples included or due to variations in other hyperparameters such as batch size which were not mentioned in the original paper
- As mentioned above, several important details and hyperparameters are lacking in the original paper. These include the batch size (n=32 seemed to be best based on the current hyperparameter search) and weight initialization (Kaiming and Xavier were chosen here

based on preliminary tests comparing them to default initializations; data not shown)

- Arterial blood pressure appeared to be the most important tracing to predict subsequent hypotensive events. Ablation studies demonstrated that models only based on ECG or EEG data are very difficult or nearly impossible to train (the training performed for the current analysis demonstrates overfitting of the training dataset for ECG and EEG tracings, likely related to the very deep model and the relatively few samples included here. this is supported by a slightly better performance [similar to what was seen in the original paper] when using a more shallow model for this ablation [data not shown]). Physiologically, it is expected that low blood pressure predicts subsequent low blood pressure events and that the ECG and especially EEG (which is a noisy tracing or brain activity) do not correlate with future hypotensive events
- The performance of the shallow model with only half as many layers is basically equivalent to the original model. However, it might be possible that the performance of the deeper model can be improved more significantly with a much larger dataset
- There was essentially no difference between using RELU and leaky RELU activations

# References

1. Jo Y-Y, Jang J-H, Kwon J, Lee H-C, Jung C-W, Byun S, Jeong H-G. Predicting intraoperative hypotension using deep learning with waveforms of arterial blood pressure, electroencephalogram, and electrocardiogram: Retrospective study. PLOS ONE 2022;17:e0272055. doi: 10.1371/journal.pone.0272055
2. Salmasi V, Maheshwari K, Yang D, Mascha EJ, Singh A, Sessler DI, Kurz A. Relationship between Intraoperative Hypotension, Defined by Either Reduction from Baseline or Absolute Thresholds, and Acute Kidney and Myocardial Injury after Noncardiac Surgery: A Retrospective Cohort Analysis. Anesthesiology 2017;126:47–65. doi: 10.1097/ALN.0000000000001432
3. Li Q, Mark RG, Clifford GD. Artificial arterial blood pressure artifact models and an evaluation of a robust blood pressure and heart rate estimator. Biomed Eng Online 2009;8:13. doi: 10.1186/1475-925X-8-13.