Heart Condition Classification based on ECG Images

CSE 472
Machine Learning Sessional

1705071 - Prantik Paul
1705074 - Md. Tanzim Azad

Supervised By
Dr. Mohammad Saifur Rahman

Problem Definition

Given sample ECG images with several number of leads , predict the human’s heart condition.

Problem Definition

Our main object is to classify various conditions of a human heart based on sample ECG images. The dataset contains five classes of images, i.e.

1. Normal Heart
2. Abnormal Heart
3. Myocardial Infarction
4. History of Myocardial Infarction
5. Covid-19 Affected

Dataset and Analysis

ECG images with several leads which are later preprocessed to fit in different architectures.

Dataset and Analysis

Initially we had 2746 ECG images of different heart conditions:

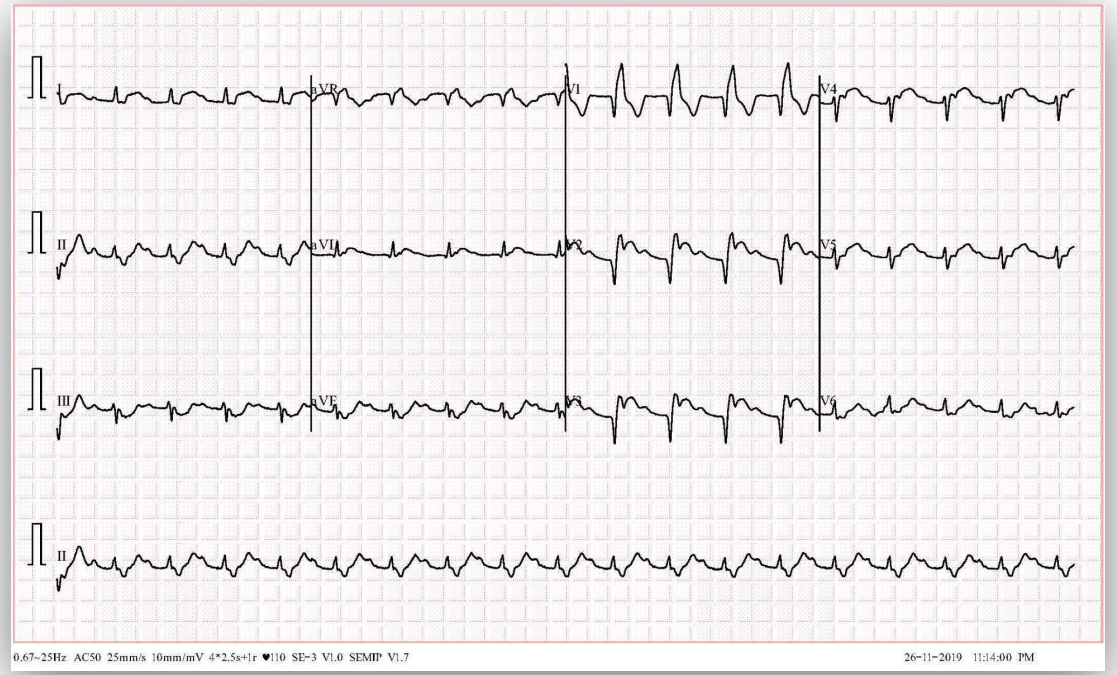
1. Normal Heart - 1143
2. Abnormal Heart - 714
3. Myocardial Infarction -313
4. History of Myocardial Infarction - 327
5. Covid-19 Affected - 250

Dataset and Analysis

Later we divided the images into multiple leads resulting in 13 different images from each image amounting to a total dataset of 35698 images:

1. Normal Heart - 14859
2. Abnormal Heart - 9282
3. Myocardial Infarction - 4069
4. History of Myocardial Infarction - 4251
5. Covid-19 Affected - 3250

A Sample ECG Image



A Sample ECG Image (Cropped Leads)

Dataset Statistics

Here is a percent wise depiction of the dataset:

1. Normal Heart - 41.6%
2. Abnormal Heart - 26%
3. Myocardial Infarction - 11.4%
4. History of Myocardial Infarction - 11.9%
5. Covid-19 Affected - 9.1%

Dataset Preprocessing

We went through the following preprocessing steps on the given images:

1. Cropping the image into 13 leads
2. Grayscaling the cropped images
3. Inverting the cropped images
4. Dilating the cropped images
5. Resizing the cropped images to 64x64

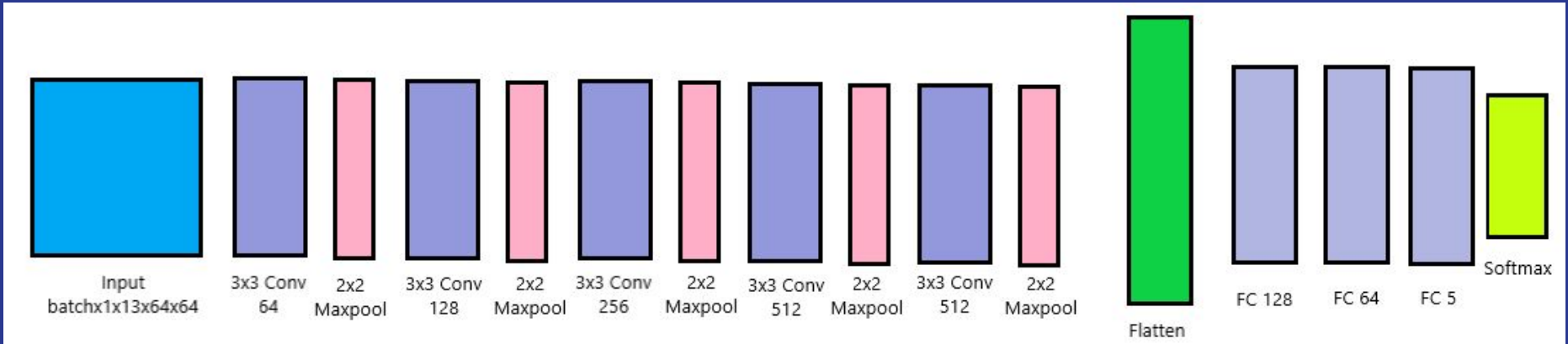
Infrastructure

Kaggle’s GPU T4x2 environment for training and testing

Proposed Architecture

We tried different architectures to approach the problems. Later we will present a comparative study among the used architectures and their performances.

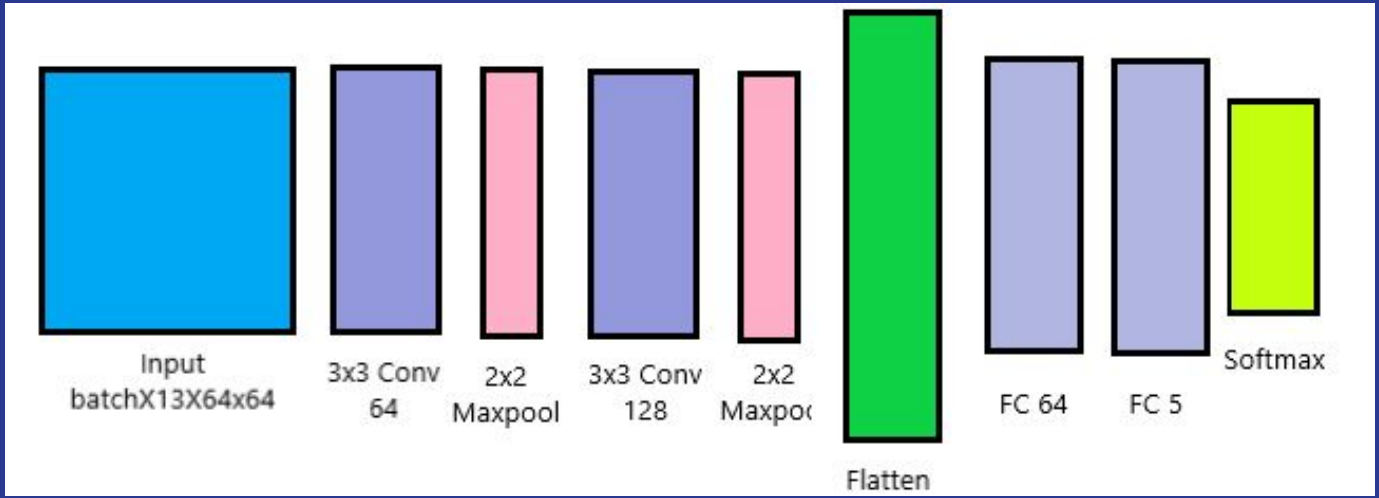
Architecture 1 (CNN 3D Network)



Architecture 1
(CNN 3D Network)

```
self.extraction_layers = nn.Sequential(
  nn.LazyConv3d(out_channels=64, kernel_size=3, padding=1),
  nn.ReLU(),
  nn.MaxPool3d(kernel_size=2, stride=2),

  nn.LazyConv3d(out_channels=128, kernel_size=3, padding=1),
  nn.ReLU(),
  nn.MaxPool3d(kernel_size=2, stride=2),

  nn.LazyConv3d(out_channels=256, kernel_size=2, padding=1),
  nn.ReLU(),
  nn.MaxPool3d(kernel_size=2, stride=2),

  nn.LazyConv3d(out_channels=512, kernel_size=2, padding=1),
  nn.ReLU(),
  nn.MaxPool3d(kernel_size=2, stride=2),

  nn.LazyConv3d(out_channels=512, kernel_size=2, padding=1),
  nn.ReLU(),
  nn.MaxPool3d(kernel_size=2, stride=2)
)

self.classification_layers = nn.Sequential(
  nn.Flatten(start_dim=1),

  nn.LazyLinear(128),
  nn.ReLU(),
  nn.LazyLinear(64),
  nn.ReLU(),
  nn.LazyLinear(5),
  nn.LogSoftmax(dim=1)
)
```

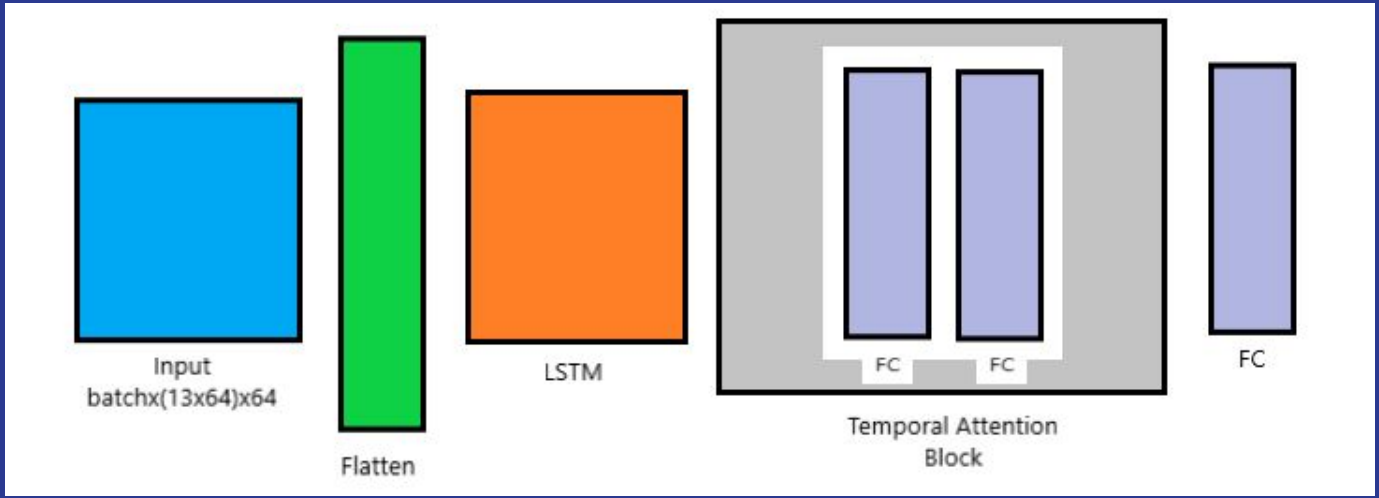
Architecture 2 (CNN 2D Network)



Architecture 2
(CNN 2D Network)

```
self.extraction_layers = nn.Sequential(
  nn.LazyConv2d(out_channels=64, kernel_size=3, padding=1),
  nn.ReLU(),
  nn.MaxPool2d(kernel_size=2, stride=2),

  nn.LazyConv2d(out_channels=128, kernel_size=3, padding=1),
  nn.ReLU(),
  nn.MaxPool2d(kernel_size=2, stride=2)
)

self.classification_layers = nn.Sequential(
  nn.Flatten(start_dim=1),
  nn.LazyLinear(64),
  nn.ReLU(),
  nn.LazyLinear(5),
  nn.LogSoftmax(dim=1)
)
```

Architecture 3 (Attention with LSTM Network)

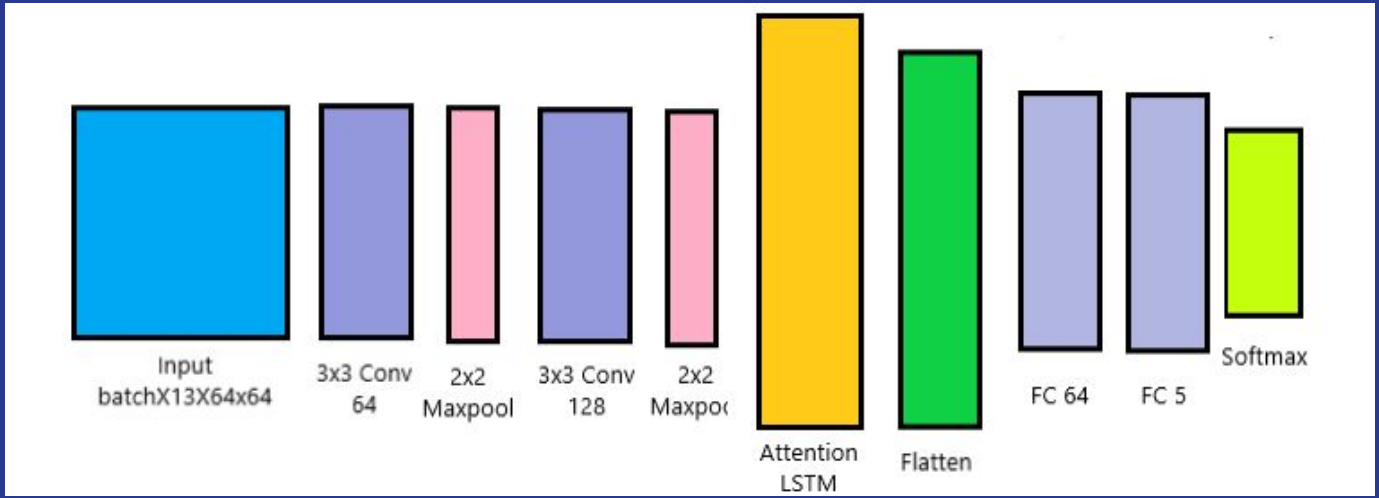


Architecture 3 (Attention with LSTM Network)

```
self.flatten = nn.Flatten(start_dim=1)
self.lstm = nn.LSTM(
  input_size=input_size,
  hidden_size=hidden_size,
  num_layers=num_layers,
  batch_first=True)
self.attn = TemporalAttn(hidden_size=hidden_size)
self.fc = nn.Linear(hidden_size, 5)
```

```
class TemporalAttn(nn.Module):
    def __init__(self, hidden_size):
      super(TemporalAttn, self).__init__()
      self.hidden_size = hidden_size
      self.fc1 = nn.Linear(self.hidden_size, self.hidden_size, bias=False)
      self.fc2 = nn.Linear(self.hidden_size*2, self.hidden_size, bias=False)
```

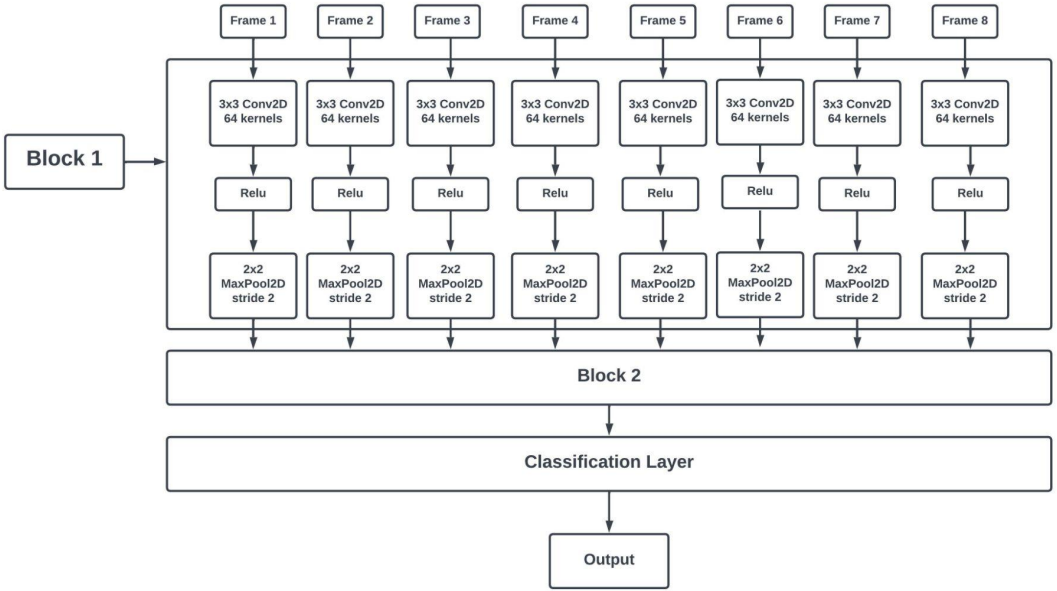
Architecture 4 (Hybrid Network)



Architecture 4 (Hybrid Network)

```
self.extraction_layers = nn.Sequential(
  nn.LazyConv2d(out_channels=64, kernel_size=3, padding=1),
  nn.ReLU(),
  nn.MaxPool2d(kernel_size=2, stride=2),

  nn.LazyConv2d(out_channels=128, kernel_size=3, padding=1),
  nn.ReLU(),
  nn.MaxPool2d(kernel_size=2, stride=2),

  nn.Flatten(start_dim=1, end_dim=2)
)

self.attn = AttnLSTM(input_size=16, hidden_size=128, num_layers=1)

self.classification_layers = nn.Sequential(
  nn.Flatten(start_dim=1),
  nn.LazyLinear(64),
  nn.ReLU(),
  nn.LazyLinear(5),
  nn.LogSoftmax(dim=1)
)
```

Architecture 5 (Time Distributed CNN Network)

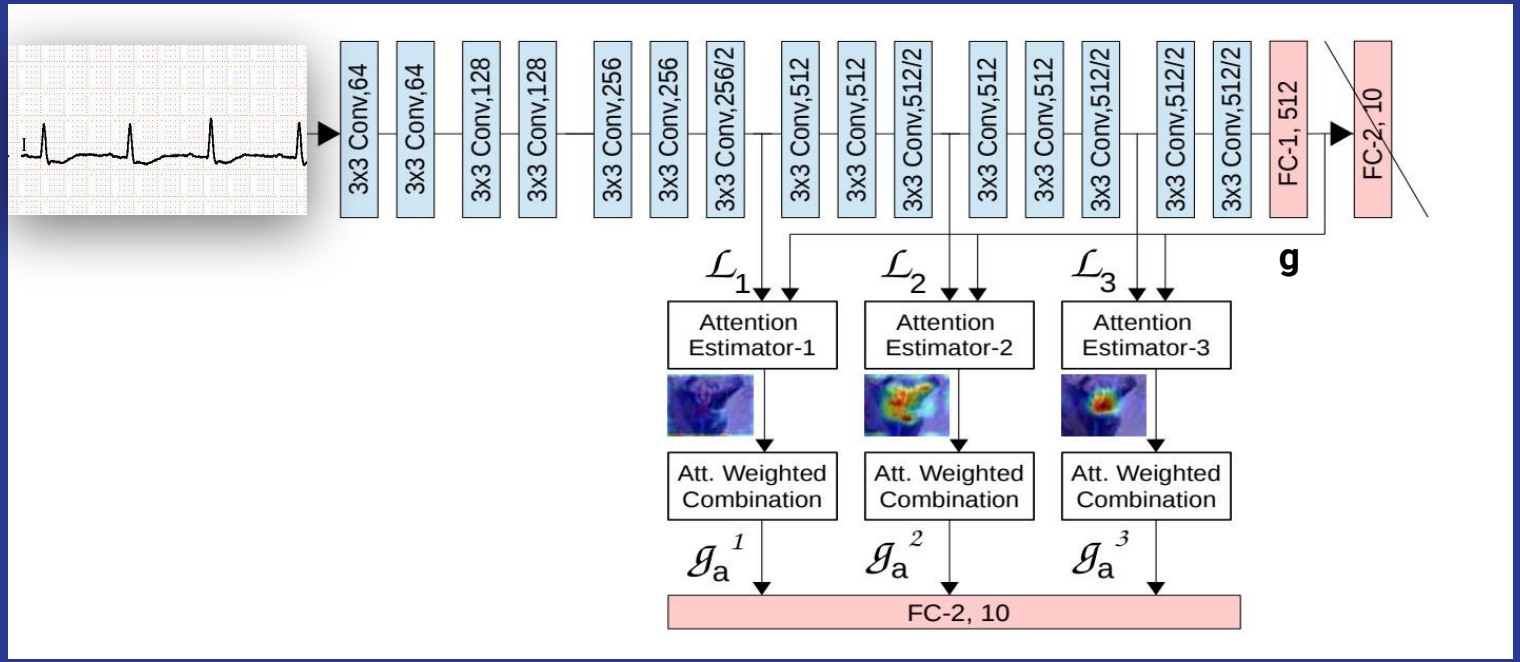


Architecture 5 (Time Distributed CNN Network)

```
self.time_steps = 8
self.extraction_layers = nn.Sequential(
  TimeDistributed(module=nn.LazyConv2d(out_channels=64, kernel_size=3, padding=1),
                  time_steps=self.time_steps),
  nn.ReLU(),
  TimeDistributed(module=nn.MaxPool2d(kernel_size=2, stride=2),
                 time_steps=self.time_steps),

  TimeDistributed(module=nn.LazyConv2d(out_channels=128, kernel_size=3, padding=1),
                  time_steps=self.time_steps),
  nn.ReLU(),
  TimeDistributed(module=nn.MaxPool2d(kernel_size=2, stride=2),
                  time_steps=self.time_steps),
)


self.classification_layers = nn.Sequential(
  nn.Flatten(start_dim=1),
  nn.LazyLinear(64),
  nn.ReLU(),
  nn.LazyLinear(5),
  nn.LogSoftmax(dim=1)
)
```

```
class TimeDistributed(nn.Module):
    def __init__(self, module, time_steps, batch_first=False):
        super(TimeDistributed, self).__init__()
        self.module = module

        self.layers = nn.ModuleList([module for i in
range(time_steps)])
        self.time_steps = time_steps
        self.batch_first = batch_first

    def forward(self, x):

        batch_size, time_steps, channels, H, W = x.size()
        output = torch.tensor([]).to("cuda:0")

        for i in range(time_steps):
            output_t = self.layers[i](x[:, i, :, :, :])
            output_t  = output_t.unsqueeze(1)
            output = torch.cat((output, output_t), 1)

        return output
```

Architecture 6 (CNN with Attention - Attention VGG)



Architecture 6 (CNN with Attention - Attention VGG)

```
# conv blocks
self.conv1 = self._make_layer(13, 64, 2)
self.conv2 = self._make_layer(64, 128, 2)
self.conv3 = self._make_layer(128, 256, 3)
self.conv4 = self._make_layer(256, 512, 3)
self.conv5 = self._make_layer(512, 512, 3)
self.conv6 = self._make_layer(512, 512, 2, pool=True)
self.dense = nn.Conv2d(in_channels=512, out_channels=512,
                       kernel_size=int(sample_size/32), padding=0, bias=True)

# attention blocks
self.attention = attention
if self.attention:
    self.projector = ProjectorBlock(256, 512)
    self.attn1 = SpatialAttn(in_features=512, normalize_attn=normalize_attn)
    self.attn2 = SpatialAttn(in_features=512, normalize_attn=normalize_attn)
    self.attn3 = SpatialAttn(in_features=512, normalize_attn=normalize_attn)

# final classification layer
if self.attention:
    self.classify = nn.Linear(in_features=512*3, out_features=num_classes,
                              bias=True)
else:
    self.classify = nn.Linear(in_features=512, out_features=num_classes,
                              bias=True)
```

```
def _make_layer(self, in_features, out_features,
                blocks, pool=False):
    layers = []
    for i in range(blocks):
        conv2d = nn.Conv2d(in_channels=in_features,
                           out_channels=out_features,
                           kernel_size=3, padding=1, bias=False)
        layers += [conv2d, nn.BatchNorm2d(out_features),
                   nn.ReLU(inplace=True)]
        in_features = out_features
        if pool:
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
    return nn.Sequential(*layers)
```

Architecture 6 (CNN with Attention - Attention VGG)

```
class SpatialAttn(nn.Module):
  def __init__(self, in_features, normalize_attn=True):
    super(SpatialAttn, self).__init__()
    self.normalize_attn = normalize_attn
    self.op = nn.Conv2d(in_channels=in_features, out_channels=1,
        kernel_size=1, padding=0, bias=False)
```

```
class ProjectorBlock(nn.Module):

  def __init__(self, in_features, out_features):

    super(ProjectorBlock, self).__init__()
    self.op = nn.Conv2d(in_channels=in_features,
                        out_channels=out_features,
                        kernel_size=1, padding=0,
                         bias=False)

  def forward(self, x):
    return self.op(x)
```

Loss Function

Multiclass Cross Entropy Loss

Loss Function and its Intuition

Multiclass Cross Entropy Loss

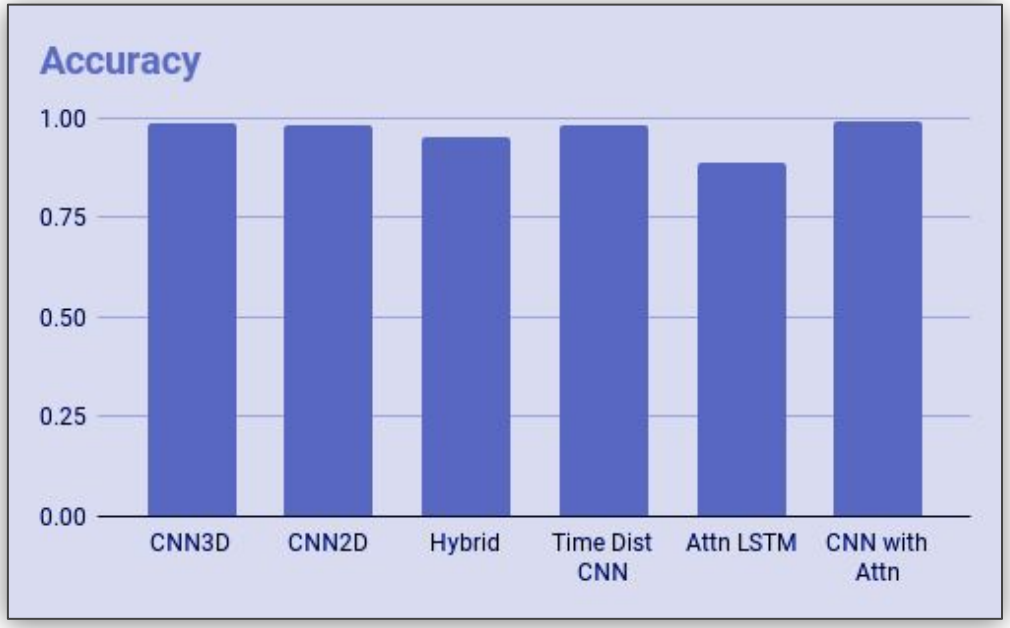
Since our objective is to classify among the five mentioned classes of heart conditions, the multiclass cross entropy loss seemed fit.

Performance Report

We calculated overall **Accuracy** and **Precision**, **Recall**, **F1 Score**, **Specificity** for each classes for each architecture.

Performance Report (Accuracy)

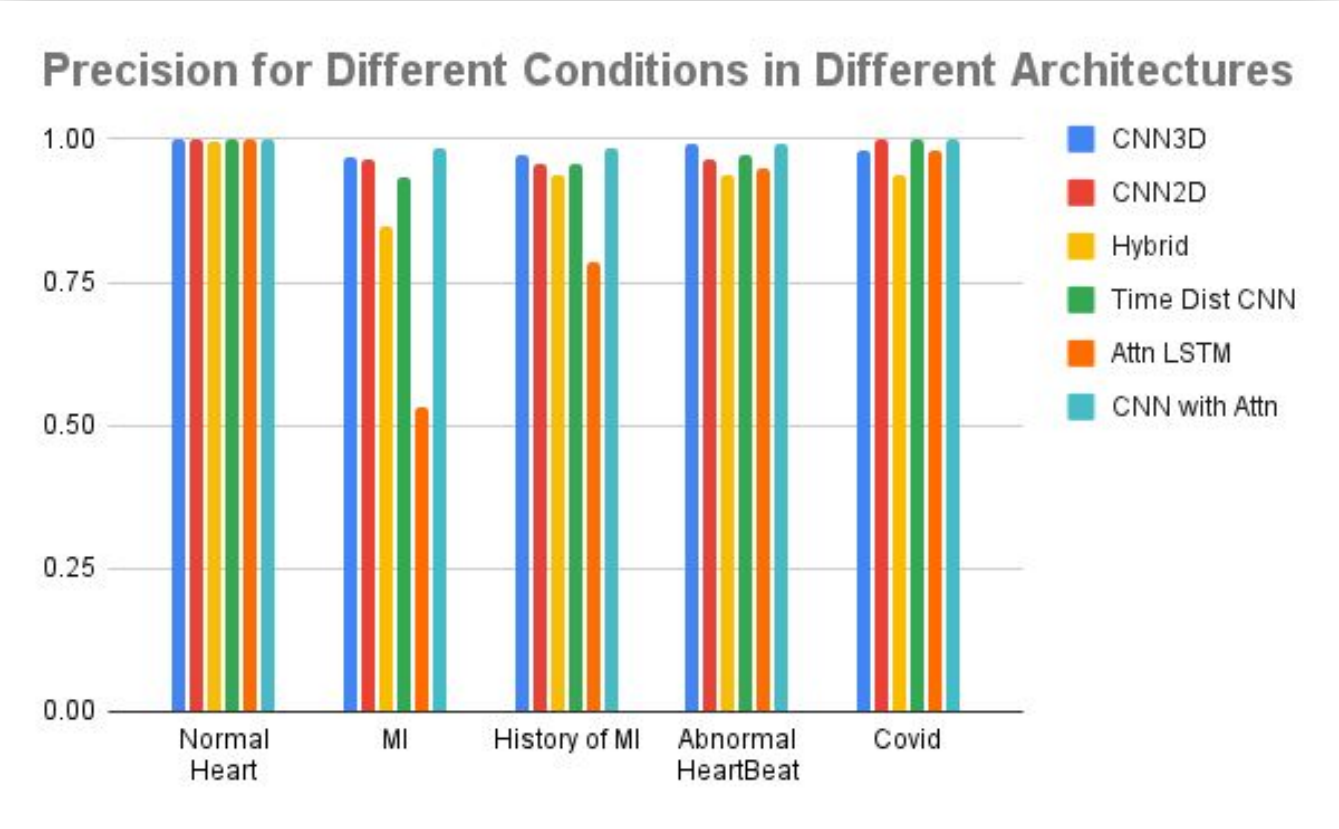
Architecture	Accuracy
CNN3D	0.98909
CNN2D	0.98182
Hybrid	0.95273
Time Dist CNN	0.98000
Attn LSTM	0.88727
CNN with Attn	0.99455



Performance Report (Precision)

Precision	Normal Heart	MI	History of MI	Abnormal HeartBeat	Covid
CNN3D	1.00000	0.96721	0.97183	0.99281	0.97872
CNN2D	1.00000	0.96491	0.95833	0.96503	1.00000
Hybrid	0.99571	0.84746	0.93846	0.93750	0.93878
Time Dist CNN	1.00000	0.93333	0.95833	0.97143	1.00000
Attn LSTM	1.00000	0.53061	0.78378	0.94853	0.97872
CNN with Attn	1.00000	0.98387	0.98571	0.99286	1.00000

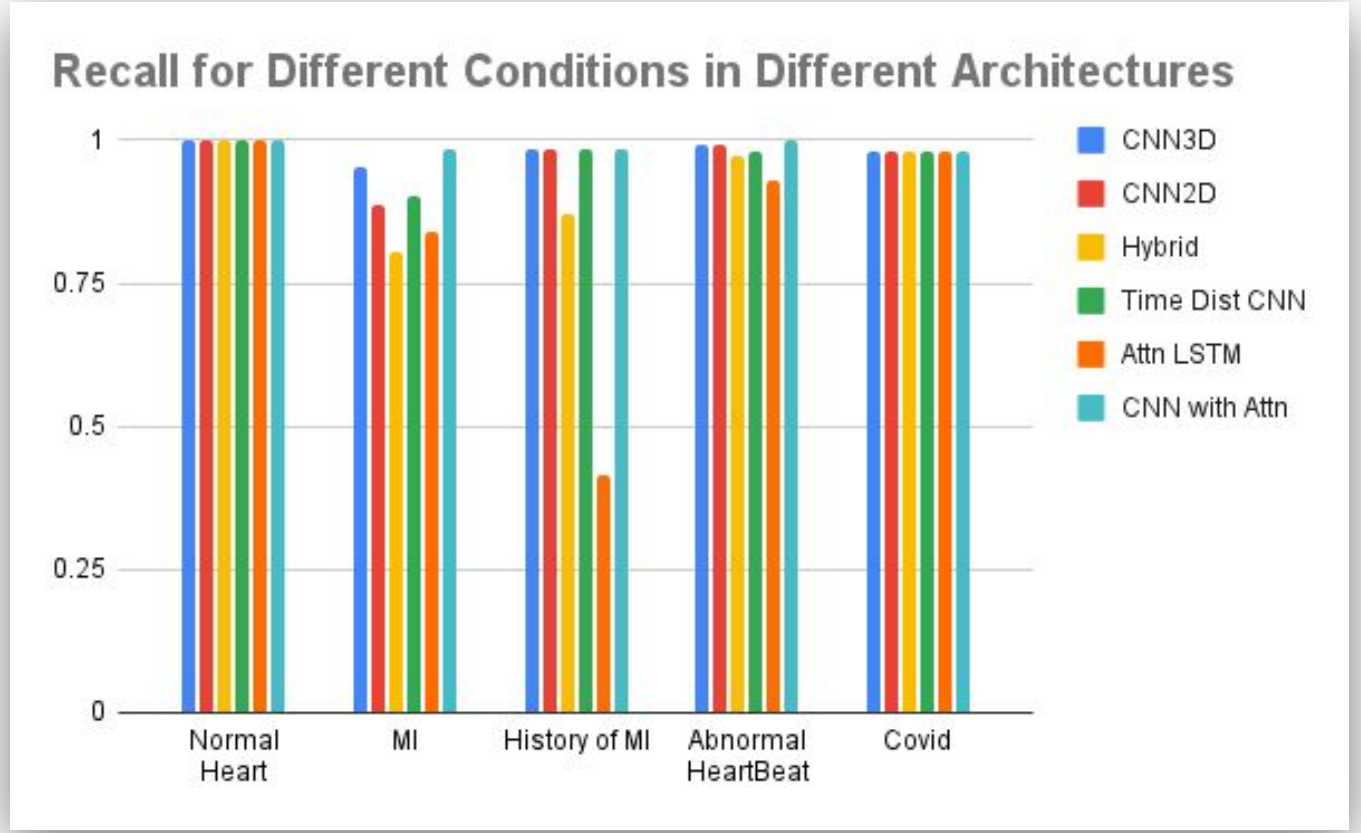
Performance Report (Precision)



Performance Report (Recall)

Recall	Normal Heart	MI	History of MI	Abnormal HeartBeat	Covid
CNN3D	1.00000	0.95161	0.98571	0.99281	0.97872
CNN2D	1.00000	0.88710	0.98571	0.99281	0.97872
Hybrid	1.00000	0.80645	0.87143	0.97122	0.97872
Time Dist CNN	1.00000	0.90323	0.98571	0.97842	0.97872
Attn LSTM	1.00000	0.83871	0.41429	0.92806	0.97872
CNN with Attn	1.00000	0.98387	0.98571	1.00000	0.97872

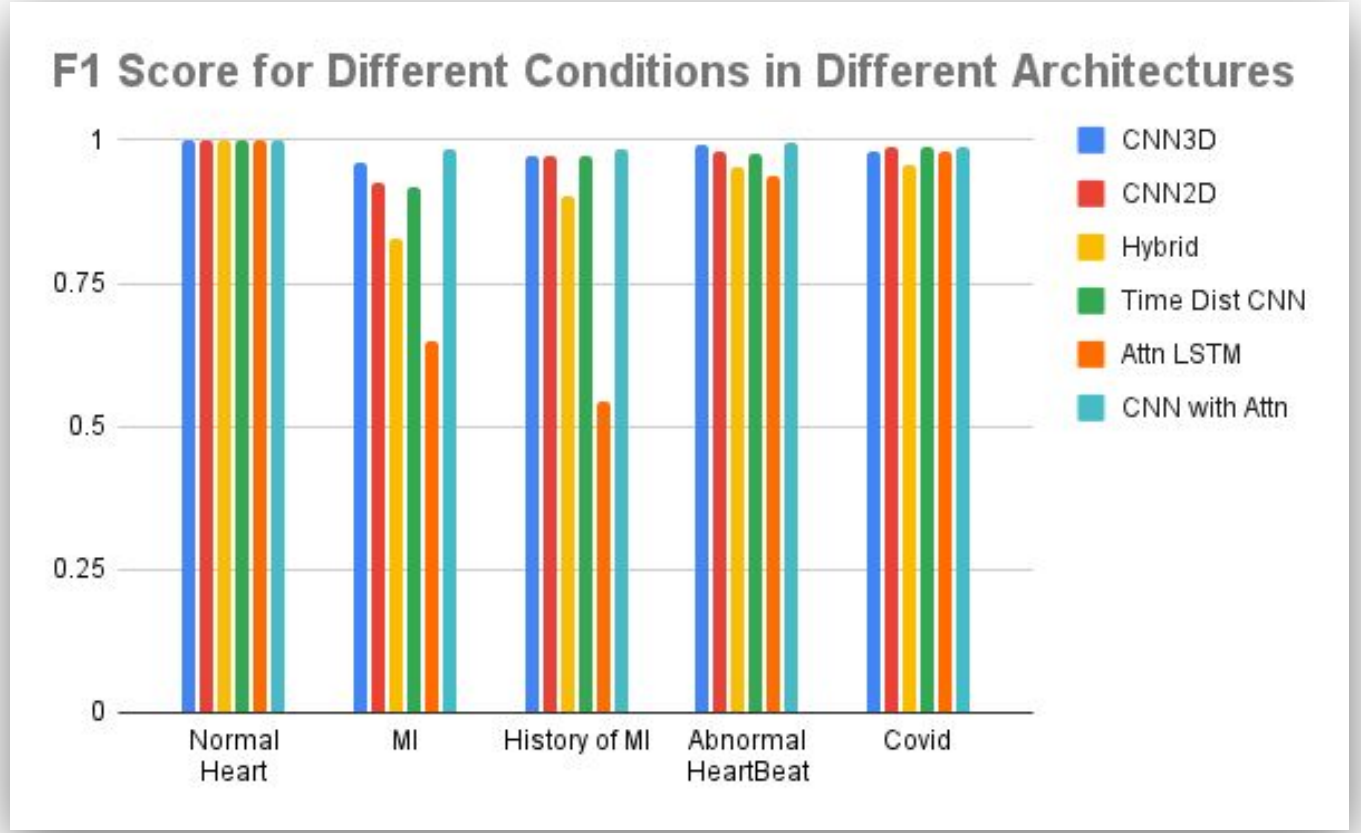
Performance Report (Recall)



Performance Report (F1 Score)

F1 Score	Normal Heart	MI	History of MI	Abnormal HeartBeat	Covid
CNN3D	1.00000	0.95935	0.97183	0.99281	0.97872
CNN2D	1.00000	0.92437	0.97183	0.97872	0.98925
Hybrid	0.99785	0.82645	0.90370	0.95406	0.95833
Time Dist CNN	1.00000	0.91803	0.97183	0.97491	0.98925
Attn LSTM	1.00000	0.65000	0.54206	0.93818	0.97872
CNN with Attn	1.00000	0.98387	0.98571	0.99642	0.98925

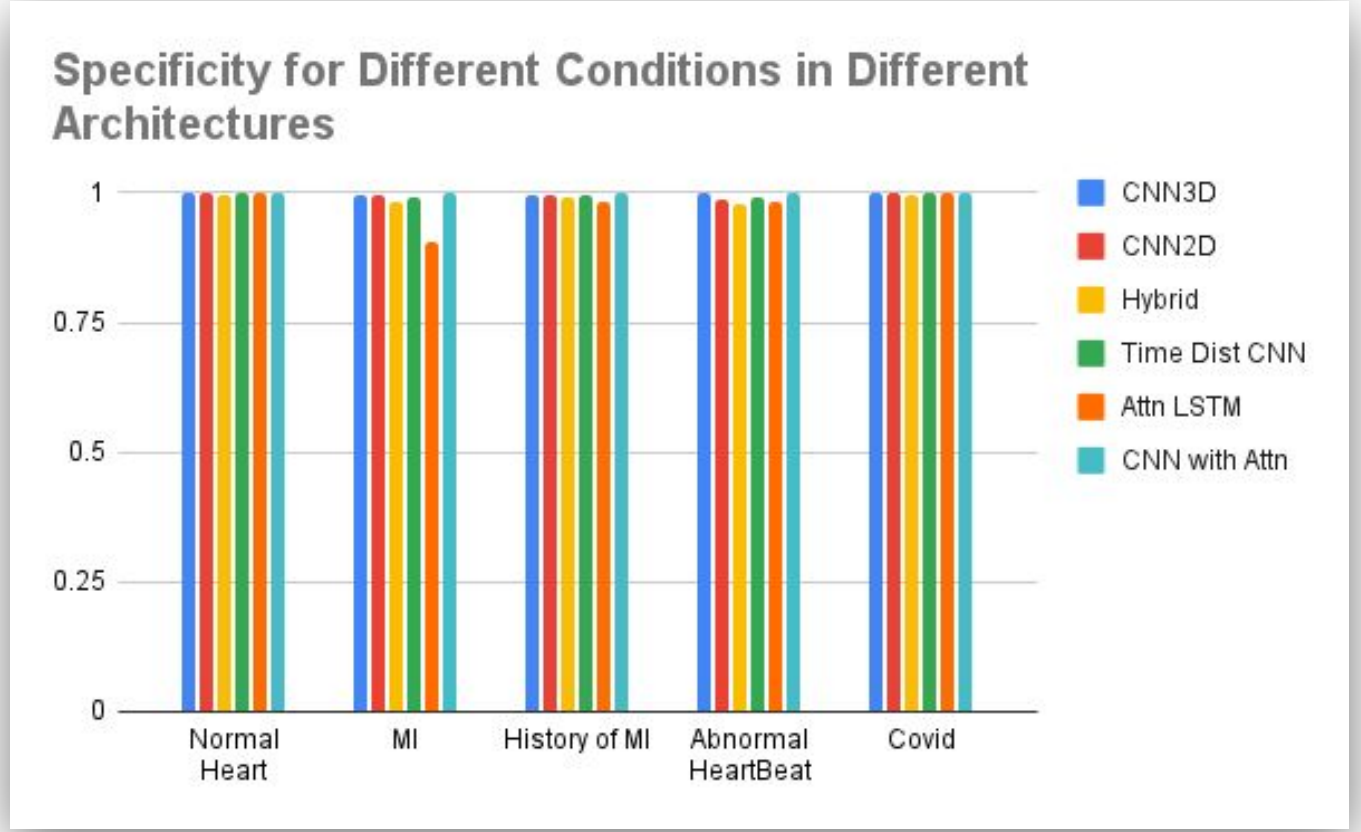
Performance Report (F1 Score)



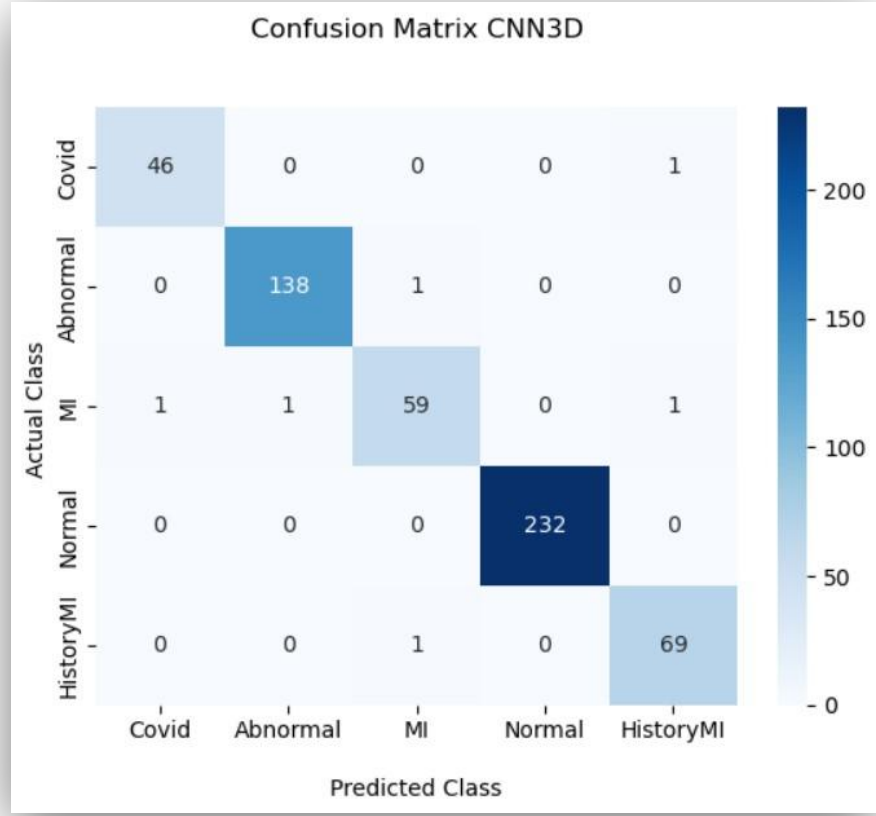
Performance Report (Specificity)

Specificity	Normal Heart	MI	History of MI	Abnormal HeartBeat	Covid
CNN3D	1.00000	0.99590	0.99583	0.99757	0.99801
CNN2D	1.00000	0.99590	0.99375	0.98783	1.00000
Hybrid	0.99686	0.98156	0.99167	0.97810	0.99404
Time Dist CNN	1.00000	0.99180	0.99375	0.99027	1.00000
Attn LSTM	1.00000	0.90574	0.98333	0.98297	0.99801
CNN with Attn	1.00000	0.99795	0.99792	0.99757	1.00000

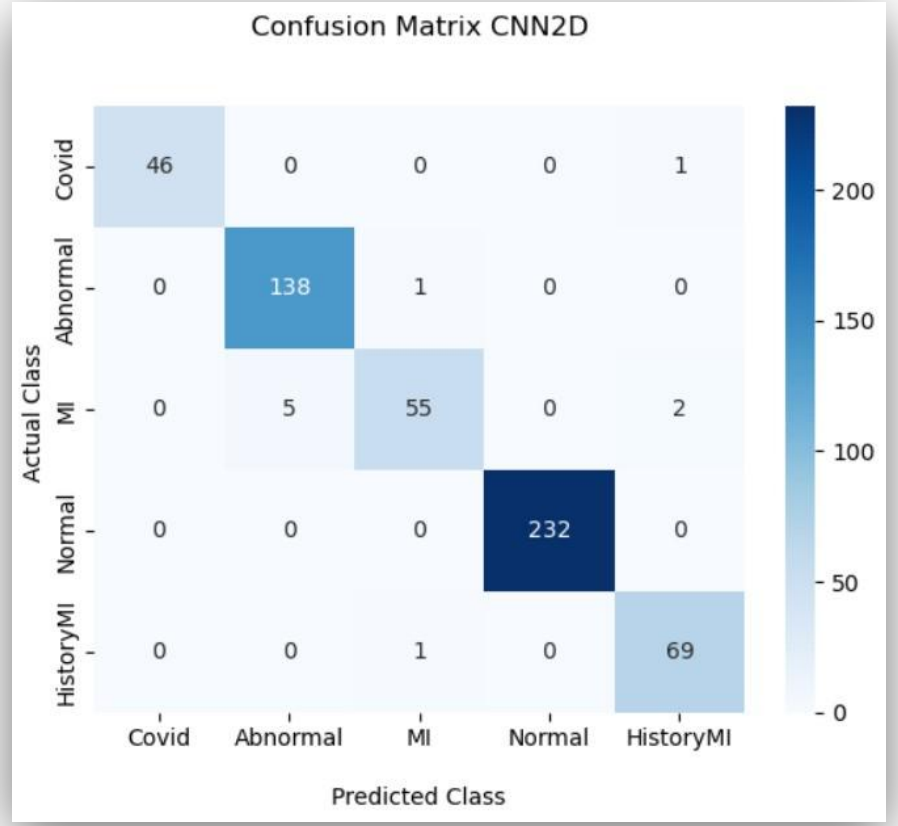
Performance Report (Specificity)



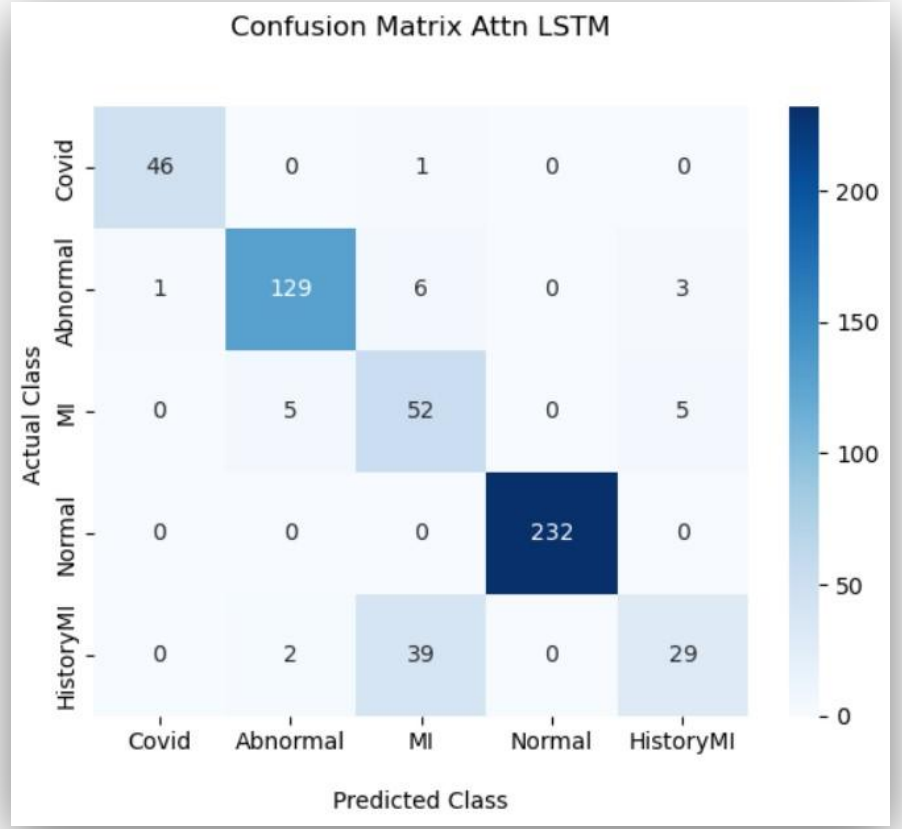
Performance Report (Confusion Matrix)



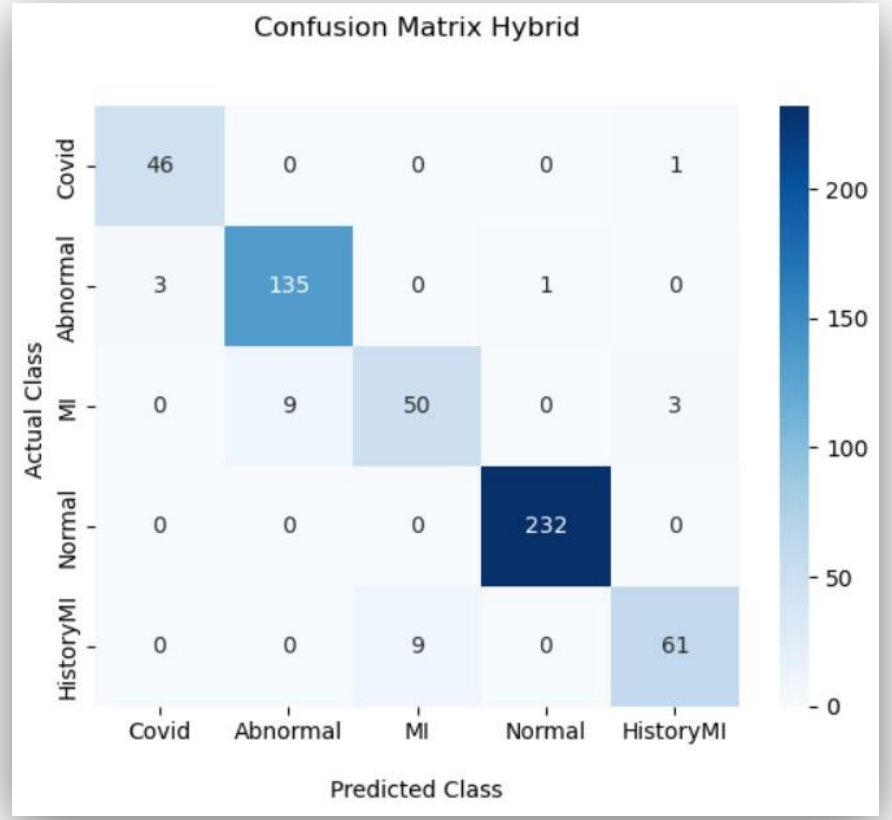
Performance Report (Confusion Matrix)



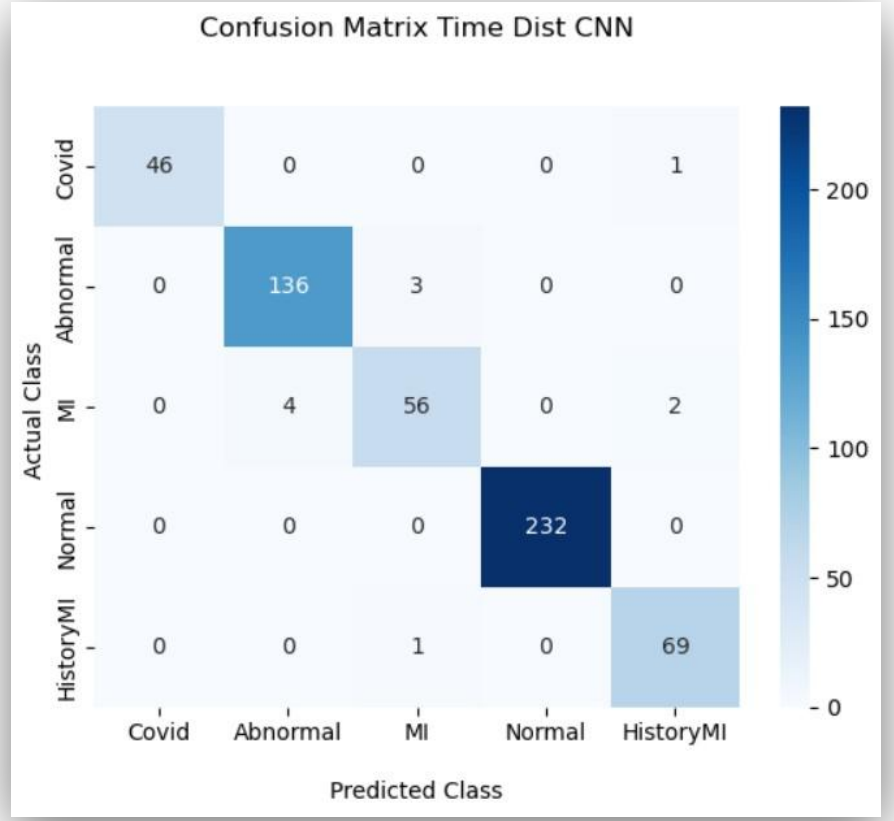
Performance Report (Confusion Matrix)



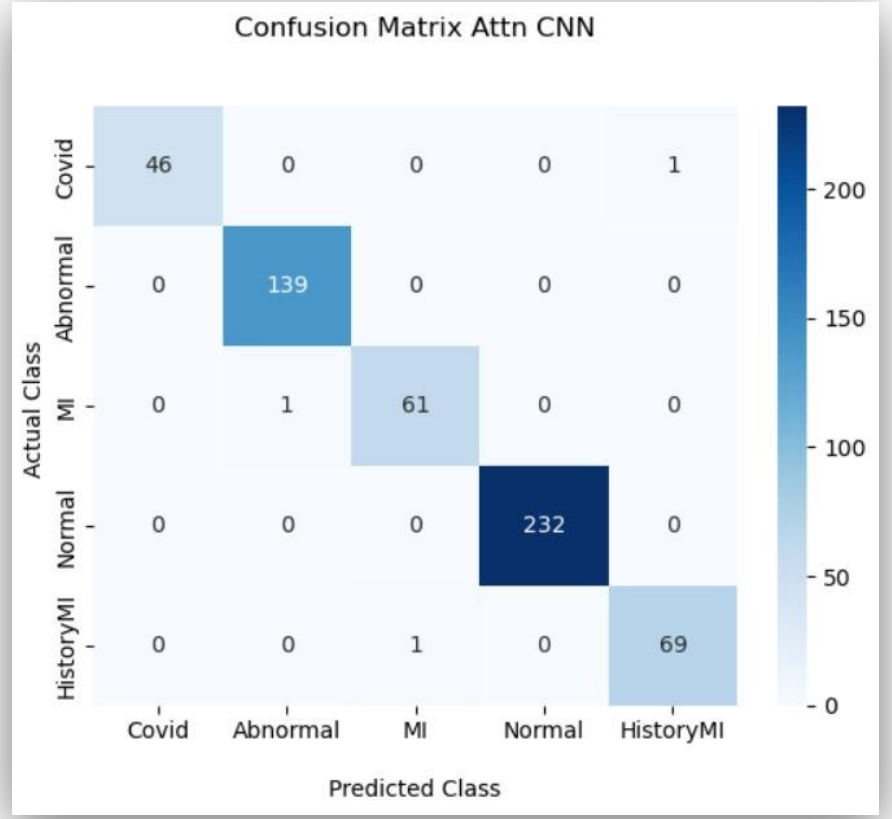
Performance Report (Confusion Matrix)



Performance Report (Confusion Matrix)



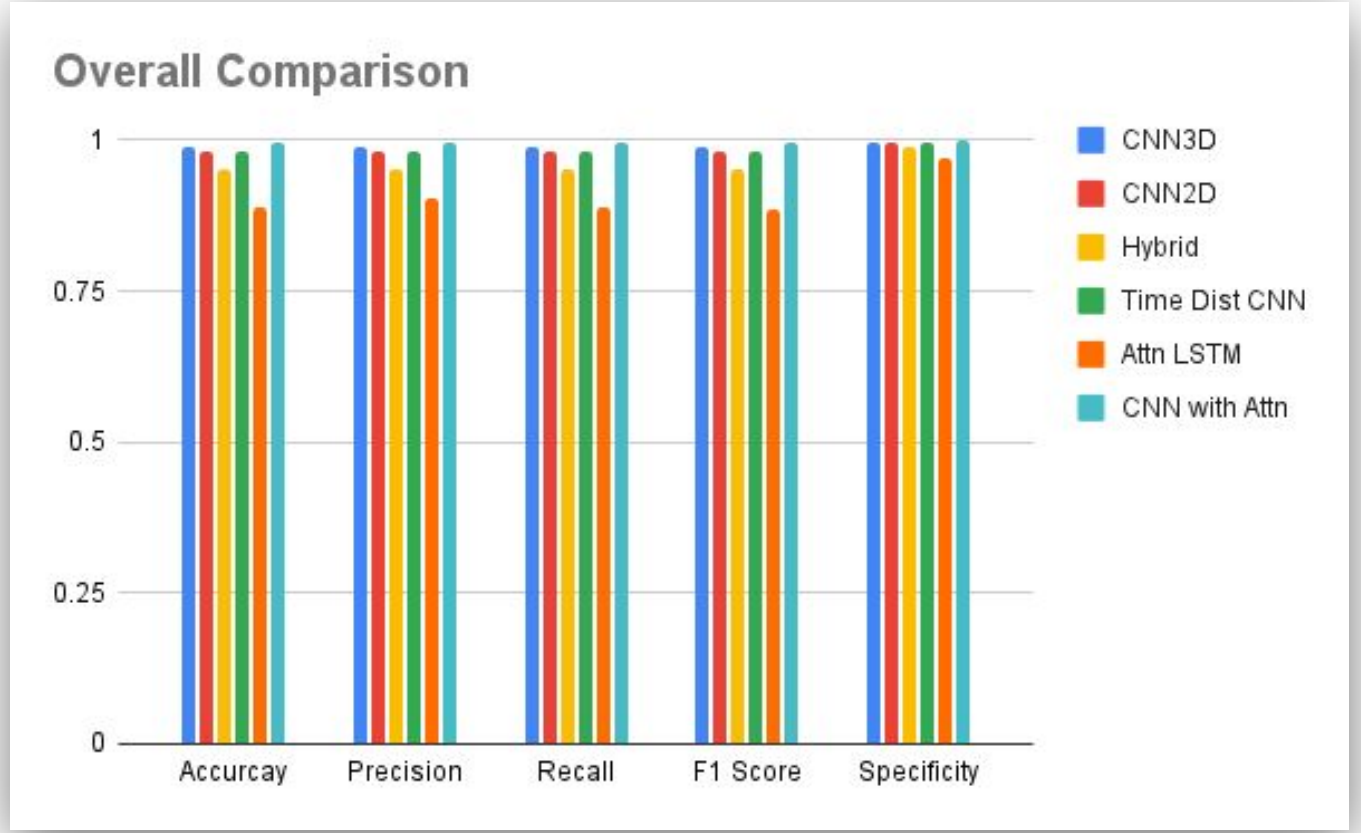
Performance Report (Confusion Matrix)



Performance Report (Decision)

Model	Accuracy	Precision	Recall	F1 Score	Specificity
CNN3D	0.98909	0.98908	0.98909	0.98907	0.99727
CNN2D	0.98182	0.98191	0.98182	0.98159	0.99545
Hybrid	0.95273	0.95213	0.95273	0.95210	0.98818
Time Dist CNN	0.98000	0.97996	0.98000	0.97992	0.99500
Attn LSTM	0.88727	0.90474	0.88727	0.88482	0.97182
CNN with Attn	0.99455	0.99456	0.99455	0.99454	0.99864

Performance Report (Decision)



Performance Report (Decision)

From overall output comparison, it can be said that CNN with Attention (**Attention VGG**) gives the most accurate results

Comparison

The state of the method we found uses ECG signals instead of direct ECG images.

Comparison with State-of-The-Art Methods

No published paper in this dataset

Model	Dataset	For Disease Classes	
		Precision	Recall
34-layer CNN	DECG/CECG	0.981/ 0.818	0.982/0.834
VGG-16	DECG/CECG	0.976/0.826	0.982/0.797
Resnet-50	DECG/CECG	0.979/0.849	0.986/0.837
FM-ECG	DECG/CECG	0.984/0.877	0.986/0.882
Proposed Model	ECG(Cardiac)	0.991	0.991

Challenges

We faced several challenges during the implementation step of this project.

Challenges

Challenge 1

Scarcity of Data

The dataset is a little scarce compared to other huge datasets in the same field. Most ECG based works focuses on signal data.

Challenge 2

Computational Limitation

Most Models use large images, such as AlexNet (224x224).

But **Kaggle** and **Google Colaboratory** fails to process such large images.

Challenge 3

Little Scope for Improvement

From our experiment it can be concluded that the data is strongly correlated. So there is little scope for improvement.

Discussion

We present the results of our experimentation with the dataset and with different architectures.

Discussion

Correlation

The dataset is strongly correlated. Thus, even a small network can show good results (almost 90%).

We had to go through rigorous experimentation to find the best architecture.

Image Dimensions

We experimented with images with different dimensions (28x28, 64x64, 96x96). Our observation was that larger the image dimensions, better the result. However, due to computational limitations, it was not possible to experiment further with larger images.

Scope of Improvement

Ensembling with Bagging classifier technique can improve the result if applied on different subset of the training data by using different seed each time.

Resources

Papers:

State-of-the-Art Models Data

LEARN TO PAY ATTENTION

Datasets:

ECG Signal COVID 500

ECG Images dataset of Cardiac and COVID-19 Patients

Thank You