ECG anomaly detection for heart health monitoring

ENM 5310: Technical Report

Submitted By

Ruchika Singh and Cyril Jose

School Of Engineering & Applied Science
University Of Pennsylvania

University Of Pennsylvania

Abstract

ECG anomaly detection for heart health monitoring

by Ruchika Singh and Cyril Jose

ECG anomaly detection is critical for early intervention and timely treatment, potentially saving lives. Machine learning models like CNNs, LSTMs, and kernel methods have shown promise in capturing features and patterns indicative of anomalies. This enables healthcare professionals to more accurately and efficiently analyze ECG signals, improving patient outcomes and scalability. Testing on datasets like PTB-XL can provide valuable insights. Automated ECG interpretation can reduce variability and errors, benefiting emergency rooms and operation theaters, as well as Holter ECG monitoring. An ML system can facilitate quick and accurate decisions for treatment, benefiting the entire medical community.

This study implements and investigates the performance of different classifiers (predominantly CNNs such as scaled down and modified versions of ResNets, GoogLeNet) on the PTB-XL dataset which is widely considered to be the largest CVD dataset.

Contents

Abstract i

Contents ii

List of Figures iv

1 Introduction 1

2 Motivation and related work 3
- 2.1 Motivation . . . 3
- 2.2 Literature review . . . 4
- 2.3 Traditional Approaches . . . 4

3 Problem statement 6
- 3.1 Mathematical Statement . . . 6
- 3.2 Dataset and Features . . . 7

4 Survey of classifiers 9
- 4.1 Google Net . . . 9
- 4.2 ResNet . . . 12
- 4.3 GoogLeNet with skip connections . . . 15
- 4.4 LSTMs . . . 17
- 4.5 Kernel Methods . . . 18

5 Results, discussions, and limitations 20
- 5.1 GoogLeNet . . . 20
  - 5.1.1 Architecture and hyperparameter details . . . 20
  - 5.1.2 Results . . . 22
  - 5.1.3 Comments . . . 22
- 5.2 ResNet . . . 24
  - 5.2.1 Architecture and hyperparameter details . . . 24
  - 5.2.2 Results . . . 26
  - 5.2.3 Comments . . . 26
- 5.3 Google Net with skip connections . . . 28
  - 5.3.1 Details . . . 28
  - 5.3.2 Results . . . 30

5.3.3 Comments 30
5.4 LSTMs 31

6 Conclusions and Future works **32**

7 References **34**

List of Figures

3.1 First 9 ECG recordings from the training set. 8

4.1 Inception block. Picture taken from [2] 10
4.2 Ordinary and PreAct ResNet blocks. Picture taken from [2] 13
4.3 Effect of skip connections. Picture taken from [2] 16

5.1 Architecture of GoogLeNet implemented. 'IB' refers to Inception Block described in previous chapters 20
5.2 Results for GoogLeNet 22
5.3 ResNet architecture implemented in Jax. Taken from [2] 24
5.4 Results for ResNet 26
5.5 One variant of GoogLeNet with skip connections implemented. 'IB' refers to Inception Block described in previous chapters 28
5.6 Results for GoogLeNet with skip connections 30

Chapter 1

Introduction

Electrocardiogram (ECG) signals are an important diagnostic tool in cardiology that measure the electrical activity of the heart. These signals are critical for detecting and diagnosing various cardiac conditions, such as arrhythmias, ischemia, and heart failure. Early detection of such anomalies can be life-saving and significantly improve patient outcomes. However, interpreting ECG signals can be a challenging task that requires significant expertise and experience. Errors or variability in interpretation can lead to incorrect diagnoses and delays in treatment. Machine learning (ML) models have shown promise in improving the accuracy and efficiency of ECG signal analysis, by automatically detecting patterns and features that are indicative of anomalies. This can benefit healthcare professionals in emergency rooms, operation theaters, and Holter ECG monitoring. By reducing variability and errors in ECG interpretation, an ML system can facilitate quick and accurate decisions for treatment, benefiting the entire medical community. In recent years, deep learning models, such as convolutional neural networks (CNNs) and long short-term memory (LSTM) networks, have shown great potential in ECG signal analysis. CNNs are particularly effective in capturing the temporal and spectral information inherent in ECG signals. This allows for the detection of specific features and patterns within the signals that

may be indicative of anomalies such as arrhythmias or ischemia. LSTMs, on the other hand, are well-suited for capturing the sequential dependencies within ECG signals. By utilizing memory cells, they can take into account previous information and use it to inform future predictions. This is particularly useful for analyzing ECG signals, which are inherently sequential in nature. In addition, kernel methods have also been explored in ECG signal analysis, as they allow for the expansion of the feature space to an infinite dimension, potentially improving classification performance.

Chapter 2

Motivation and related work

2.1 Motivation

ECG signal analysis is a crucial task in the diagnosis and treatment of cardiovascular diseases. However, the manual interpretation of ECG signals is time-consuming, labor-intensive, and prone to human error. An ML system that can accurately and efficiently detect anomalies in ECG signals can potentially improve patient outcomes, reduce healthcare costs, and enhance the quality of care. Furthermore, as the use of ECG signals continues to grow, there is a need for automated systems that can process the large volumes of data generated. An ML system that can analyze ECG signals can help healthcare professionals make informed decisions, leading to improved patient outcomes. Therefore, the development of ML models that can accurately detect anomalies in ECG signals is a critical area of research, and this project aims to explore the performance of various ML models on the PTB-XL dataset.

2.2 Literature review

Several studies have investigated the use of ML models for ECG signal analysis. For instance, in (Acharya et al., 2017), a convolutional neural network (CNN) was used to classify ECG signals into various arrhythmias. The study found that the CNN model outperformed other ML models such as support vector machines and decision trees.

In another study (Dey et al., 2020), a deep learning-based approach was used for the automated diagnosis of cardiac arrhythmias. The proposed model outperformed other state-of-the-art methods for ECG signal classification.

In (Rajpurkar et al., 2017), a large-scale dataset of ECG recordings, called the PTB-XL dataset, was introduced. This dataset contains over 21,000 ECG recordings from different patient groups, including healthy individuals and those with various cardiac conditions. Several ML models have been tested on this dataset, including CNNs, recurrent neural networks (RNNs), and support vector machines.

In a recent study (Gupta et al., 2021), a hybrid deep learning architecture was proposed for the classification of ECG signals. The model consists of a CNN and a long short-term memory (LSTM) network, which can capture both spatial and temporal features of the ECG signals. The proposed model achieved promising results on the PTB-XL dataset.

2.3 Traditional Approaches

Traditional approaches for ECG classification typically rely on handcrafted features and classifiers such as support vector machines (SVMs), random forests, and artificial neural networks (ANNs). These approaches involve extracting features from ECG signals that are believed to be important in diagnosing certain conditions. For example, some features used in ECG classification include

the QRS complex duration, the PR interval duration, the amplitude of the ST segment, and the QT interval duration.

Once the features have been extracted, a classifier is trained to classify the ECG signals into different categories. SVMs, for example, work by finding a hyperplane that separates the different categories in the feature space. Random forests are an ensemble method that constructs multiple decision trees and combines their predictions to improve the overall classification accuracy. ANNs use a layered architecture of artificial neurons to learn the non-linear relationships between the features and the categories.

While these traditional approaches have been successful in ECG classification, they have several limitations. First, they require extensive feature engineering, which can be time-consuming and requires expertise in signal processing. Additionally, the features selected may not be the most informative or may miss subtle patterns in the data. Second, traditional classifiers are limited in their ability to capture complex patterns in the data, especially when the relationships between the features and the categories are non-linear. Finally, the performance of these classifiers may vary depending on the dataset and the specific task, which makes them less generalizable.

Recent advances in deep learning have shown promise in addressing some of these limitations and have led to state-of-the-art results in ECG classification.

Chapter 3

Problem statement

3.1 Mathematical Statement

The goal of the ECG classification problem is to develop an algorithm that can accurately classify ECG recordings into their respective diagnostic categories. Given a dataset of ECG recordings, the task is to learn a mapping function that can map an input ECG recording to one of the diagnostic categories. Specifically, the problem can be stated mathematically as follows:

Given a dataset: $D = (\mathbf{x}i, y_i)i = 1^N$ where $\mathbf{x}_i \in \mathbb{R}^{d_1 \times d_2}$ represents an ECG recording and $y_i \in 0, 1, \ldots, M-1$ represents its corresponding diagnostic category, the goal is to learn a mapping function:

$f_\theta(\mathbf{x}) : \mathbf{x} \rightarrow y$

where θ represents the parameters of the classifier that need to be optimized. The objective is to find the optimal θ that minimizes the classification error on a holdout dataset.

The ultimate goal of this project is to improve the accuracy and efficiency of ECG anomaly detection, which can benefit healthcare professionals in emergency rooms, operation theaters, and Holter ECG monitoring. By reducing variability and errors in ECG interpretation, an ML

system can facilitate quick and accurate decisions for treatment, benefiting the entire medical community. The lack of proper quantification of ML and deep learning algorithms' performance on the PTB-XL dataset adds to the challenge, and the work done by [2] where they developed 'xresnet1d101' will be used as the current state of the art for comparison.

3.2 Dataset and Features

The dataset comprises 21799 clinical 12-lead ECG records of 10 seconds length from 18869 patients, where 52% are male and 48% are female with ages covering the whole range from 0 to 95 years (median 62 and interquartile range of 22). The value of the dataset results from the comprehensive collection of many different co-occurring pathologies, but also from a large proportion of healthy control samples. The distribution of diagnosis is as follows, where we restrict for simplicity to diagnostic statements aggregated into superclasses (note: sum of statements exceeds the number of records because of potentially multiple labels per record):

Table 3.1: Distribution of records in the dataset

Superclass	Records	Description
NORM	9514	Normal ECG
MI	5469	Myocardial Infarction
STTC	5235	ST/T Change
CD	4898	Conduction Disturbance
HYP	2649	Hypertrophy

The following figure is a visualization of the ECG waveforms from the training dataset. The first 9 recordings are visualized and the title of each subfigure indicates the superclass it belongs to.

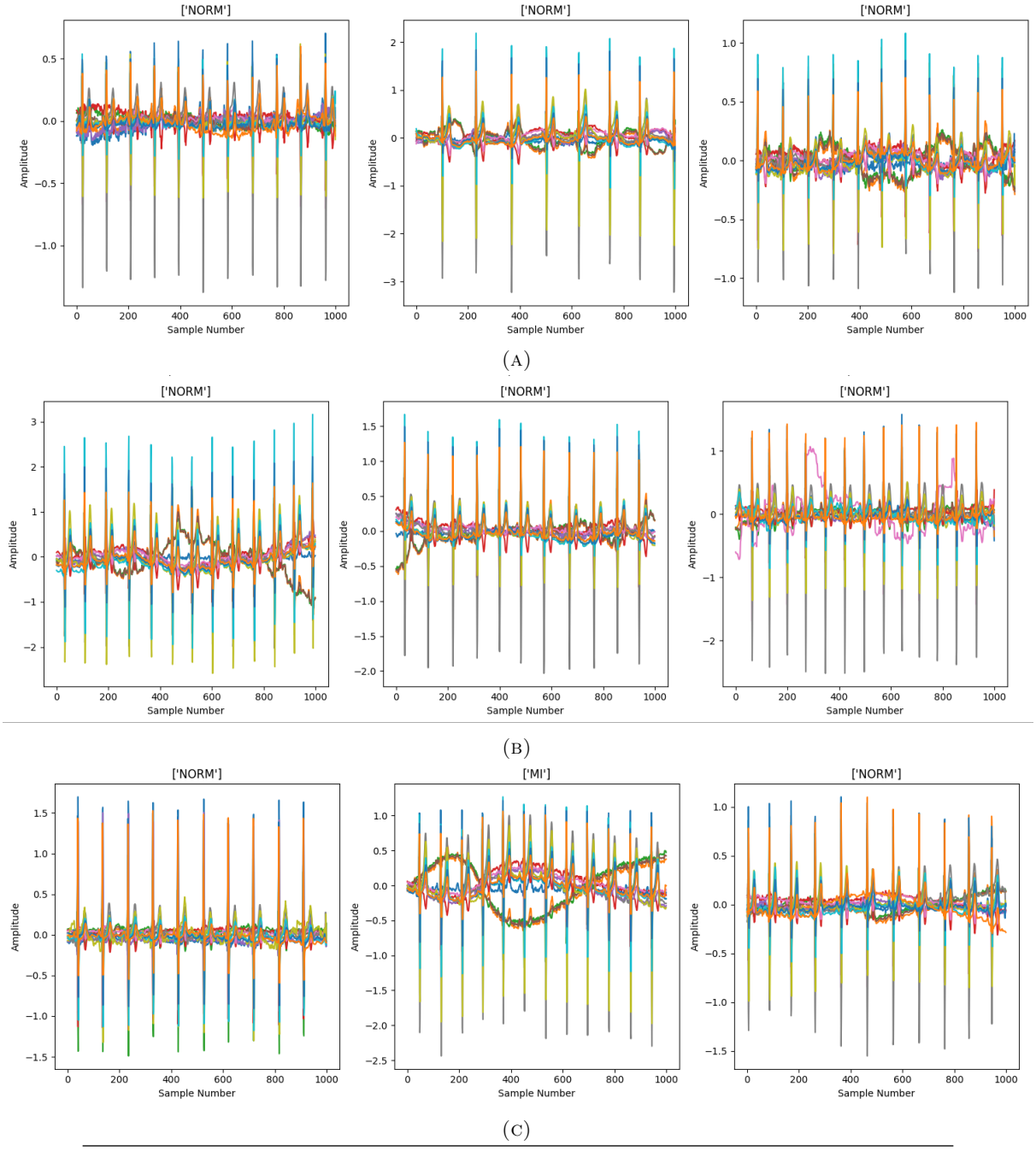


FIGURE 3.1: First 9 ECG recordings from the training set. The title of each subplot is the diagnostic superclass of the recording.

Chapter 4

Survey of classifiers

4.1 Google Net

To address the problem statement, we propose to use a scaled down version of the GoogLeNet architecture, which is a deep convolutional neural network that was designed specifically for image classification tasks. GoogLeNet is known for its accuracy and efficiency, and it achieved state-of-the-art results on the ImageNet dataset when it was introduced in 2014.

The inception block is a module used in deep convolutional neural networks for image classification. It applies four different convolutional operations on the same input feature map. These operations are a 1x1 convolution, a 3x3 convolution, a 5x5 convolution, and a max pooling operation.

The use of these different convolutions with varying receptive fields allows the network to capture features at different scales and extract more diverse features from the input data. This, in turn, improves the network's ability to recognize objects and patterns in images.

Although using only 5x5 convolution could theoretically be more powerful, it would be computationally expensive and prone to overfitting. Therefore, the inception block provides a more

efficient and effective way to process the input data. The overall inception block is visualized below:

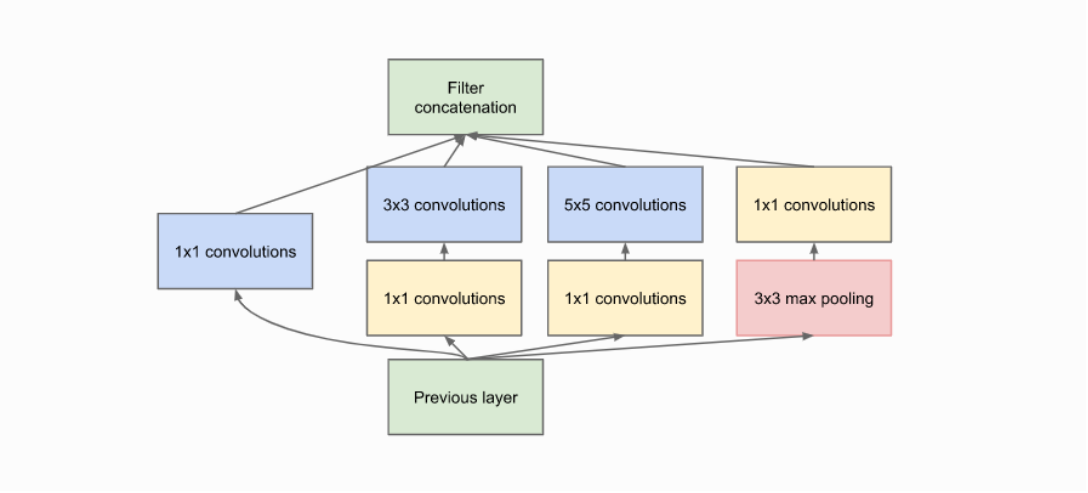


FIGURE 4.1: Inception block. Picture taken from [2]

The model is defined as a subclass of nn.Module with a call method that accepts an input tensor and returns a probability distribution over the output classes. The call method first applies a 2D convolutional layer with 64 output channels, followed by batch normalization and the ReLU activation function. Then, it applies several inception blocks that use 1x1, 3x3, and 5x5 convolutional filters to extract features from the input tensor at different spatial resolutions. The number of output channels for each filter size is determined by the c_{red} and c_{out} dictionaries, which specify the number of channels for the reduced and output dimensions, respectively. The output of each block is passed through batch normalization and the activation function specified during the model instantiation.

The output of the final inception block is averaged over the spatial dimensions, flattened, and passed through a fully connected layer with the number of output classes. Finally, the output of the fully connected layer is transformed into a probability distribution over the classes using the softmax function.

The InceptionBlock class is defined as a subclass of nn.Module and represents a single inception block in the GoogleNet model. The call method of InceptionBlock takes an input tensor and

applies 1x1, 3x3, and 5x5 convolutional filters to extract features at different spatial resolutions, followed by concatenation along the channel dimension. The output of the concatenated tensor is passed through batch normalization and the activation function specified during the block instantiation.

The GoogleNet and InceptionBlock classes use the nn.compact decorator to specify the forward pass of the model and block, respectively. Additionally, the model and block use the Kaiming normal initialization scheme for the convolutional layers and batch normalization to improve the stability and convergence of the training process. Finally, we will deploy the trained network to predict the labels of new, unseen waveform that are relevant to the problem statement. We will measure the network's performance on these waveform datasets using standard metrics such as accuracy , F1 score , and confusion matrix.

4.2 ResNet

Another model that we implemented and trained is ResNet , a deep convolutional neural network architecture that is highly effective in waveform classification tasks. The architecture is based on the concept of residual connections, which enables the network to effectively propagate gradients through the network during backpropagation. The ResNet paper is one of the most cited AI papers and has been the foundation for neural networks with more than 1,000 layers.
The non-linear mapping is a sequence of neural network modules like convolutions, activation functions, and normalizations. By using residual connections, the network can learn residual mappings between different layers. The bias towards the identity matrix in residual connections guarantees stable gradient propagation, making the network less sensitive to the layers' depth.

Residual connections are a simple yet effective technique to improve gradient propagation in deep neural networks. Instead of directly modeling a mapping function F(x), we model the residual mapping function $x_{l+1} = F(x) + x_l$, where x is the input and F(x) is a sequence of neural network modules such as convolutions, activation functions, and normalizations. By backpropagating through the residual connections, the gradient can propagate more stably through the network.

To achieve this, the residual connections are designed to have a bias towards the identity matrix, which helps to mitigate the problem of vanishing gradients that can occur when deeper networks are trained. This bias makes the gradient propagation less sensitive to the function F(x) itself.

$$\frac{\partial x_{l+1}}{\partial x_l} = \mathrm{I} + \frac{\partial F(x_l)}{\partial x_l}$$

There are many variants of ResNet proposed, each with their own modifications to the function

F(x) or operations applied on the sum.For our project we focused on the implementation of two of them :the original ResNet block, and the Pre-Activation ResNet block. The two blocks have been visualized below(figure credit - He et al.):

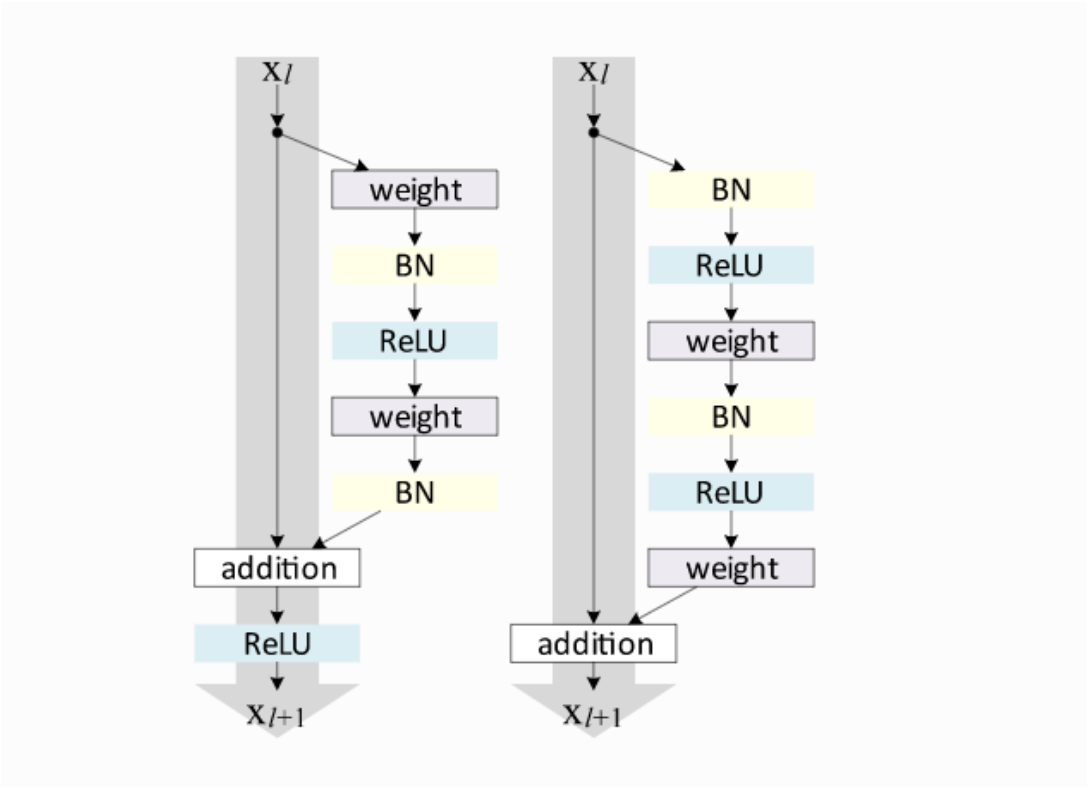


FIGURE 4.2: Ordinary and PreAct ResNet blocks. Picture taken from [2]

To implement the original ResNet block, we need to handle the case where we want to reduce the waveform dimensions in terms of width and height.Therefore, we need to change the dimensionality as well before adding to . The original implementation uses an identity mapping with stride 2 and padded additional feature dimensions with 0. However, the more common implementation uses a 1x1 convolution with stride 2 as it allows us to change the feature dimensionality while being efficient in parameter and computation cost.

The code for the ResNet block is relatively simple, and it's defined as a subclass of nn.Module with a call method that accepts an input tensor and returns a tensor. The call method applies a sequence of convolutional, batch normalization, and activation layers with residual connections.

The residual connection is added to the output of the sequence of convolutional layers, followed by another activation function. The network uses the nn.compact decorator to specify the forward pass of the block and uses the Kaiming normal initialization scheme for the convolutional layers and batch normalization to improve the stability and convergence of the training process.

4.3 GoogLeNet with skip connections

In the call method of the GoogleNet class, the inception blocks list contains instances of the InceptionBlock class and a lambda function.

In the for loop over the inception blocks list, the variable skip is being used to store the output tensor of the previous inception block. When the current block is not an instance of the InceptionBlock class, it is a max pooling layer and the output tensor is computed by applying the max pooling operation on the input tensor x.

When the current block is an instance of the InceptionBlock class, the x tensor is passed as input to the call method of the inception block and the output tensor is stored in x. Before passing the x tensor to the next block, the skip tensor is added to it and the result is assigned back to x.

In this way, the output tensor of the current inception block is being added to the input tensor of the next inception block. This is called a "skip connection" or a "residual connection", and it allows the gradients to flow more easily through the network, which can help to avoid the vanishing gradient problem.

The skip connection is being implemented specifically for the second, fifth, and ninth inception blocks in the inception blocks list, as indicated by the condition if $count == 0$ or $count == 1$ or $count == 2$. This means that the first two inception blocks are followed by a max pooling layer without a skip connection, and all subsequent inception blocks are connected via skip connections.

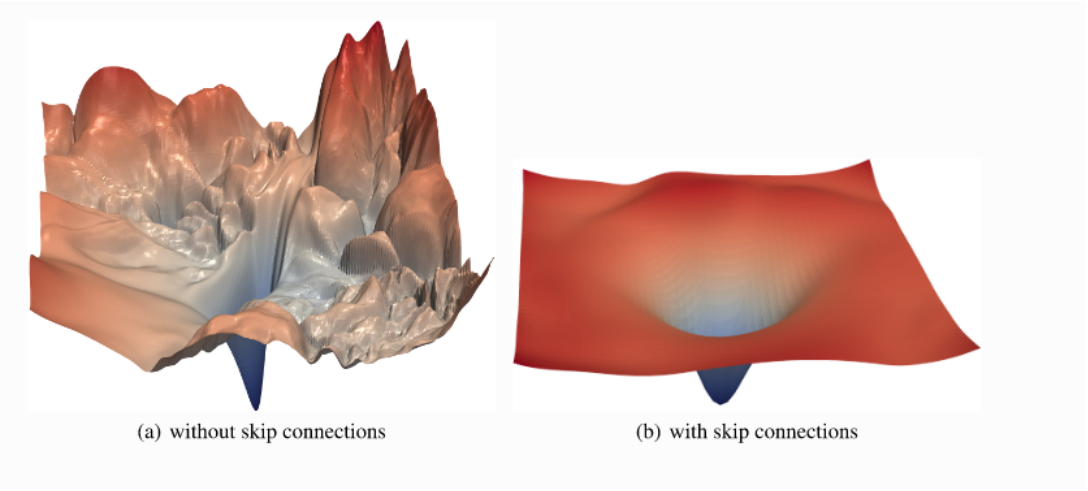


FIGURE 4.3: Effect of skip connections. Picture taken from [2]

4.4 LSTMs

An LSTM cell has three gates: the input gate (i_t), the forget gate (f_t), and the output gate (o_t). These gates control the flow of information into and out of the cell. The equations for these gates are:

$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i)$

$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$

$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)$

Here, x_t is the input at time t, h_{t-1} is the hidden state at time $t-1$, σ is the sigmoid activation function, and W and b are the weight matrices and bias vectors of the model.

The cell state c_t is updated based on the input, forget, and output gates, as well as a candidate value $\tilde{c}_t$:

$\tilde{c}_t = \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c)$

$c_t = f_t c_{t-1} + i_t \tilde{c}_t$

Finally, the output h_t is computed using the updated cell state and output gate activation:

$h_t = o_t \tanh(c_t)$

This LSTM architecture allows for capturing long-term dependencies in sequences, which makes it a useful tool for tasks such as ECG classification.

4.5 Kernel Methods

Kernel methods are a family of machine learning algorithms that can be used for ECG classification. These methods work by transforming the input data into a high-dimensional feature space, where it may be easier to separate the different classes. One popular kernel method is the Support Vector Machine (SVM) algorithm, which aims to find the hyperplane that maximally separates the data points in the feature space.

Given a training set of n ECG recordings, $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where x_i is the ith ECG recording and y_i is its corresponding diagnostic label, we can use an SVM to learn a decision function $f(x)$ that maps an ECG recording to its predicted label. The decision function takes the form:

$$f(x) = \text{sign}(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b), \tag{4.1}$$

where α_i are the SVM coefficients, b is the bias term, and $K(x_i, x)$ is the kernel function, which computes the similarity between two ECG recordings. In this case, we can use the Radial Basis Function (RBF) kernel:

$$K(x_i, x) = \exp(-\gamma ||x_i - x||^2), \tag{4.2}$$

where γ is a hyperparameter that controls the width of the Gaussian kernel.

The SVM algorithm learns the SVM coefficients α_i and the bias term b by solving the following optimization problem:

$$\begin{aligned} \underset{\alpha_i, b}{\text{minimize}} \quad & \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^{n} \alpha_i y_i = 0, \end{aligned} \tag{4.3}$$

where C is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the classification error.

Once the SVM has been trained, we can use it to predict the labels of new ECG recordings by simply computing the decision function for each recording. Kernel methods like SVMs can be effective for ECG classification, especially when the data has a complex, non-linear structure that is difficult to capture with linear models.

Chapter 5

Results, discussions, and limitations

5.1 GoogLeNet

5.1.1 Architecture and hyperparameter details

The following figure represents the architecture implemented in Jax. 'IB' refers to inception block which was described previously. There are batch normalization modules within each inception block as well.

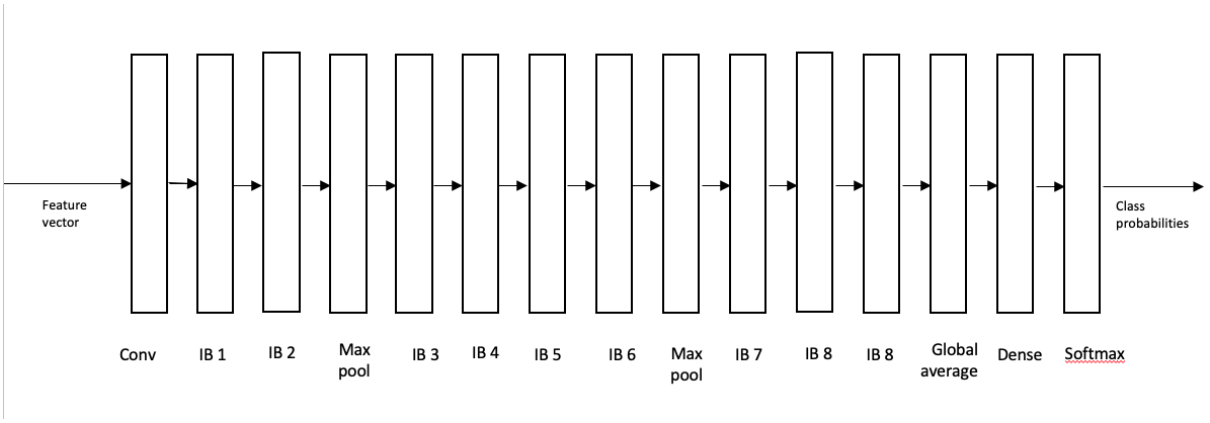


FIGURE 5.1: Architecture of GoogLeNet implemented. 'IB' refers to Inception Block described in previous chapters

The following table is a breakdown of all the parameters used for each block in the architecture.

Layer Type	**Details**
Convolution	Kernel: (3,3), No. of Filters: 64, Activation: ReLU
Batch Normalization	Momentum: 0.9, Epsilon: 0.001
Inception Block 1	Reducing Channels: 3x3-32, 5x5-16 ;Output Channels: 1x1-16, 3x3-32, 5x5-8, Max-8
Inception Block 2	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-24, 3x3-48, 5x5-12, Max-12
Max Pooling	Pooling Size: (3,3), Strides: (2,2)
Inception Block 3	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-24, 3x3-48, 5x5-12, Max-12
Inception Block 4	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-16, 3x3-48, 5x5-16, Max-16
Inception Block 5	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-16, 3x3-48, 5x5-16, Max-16
Inception Block 6	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-32, 3x3-48, 5x5-24, Max-24
Max Pooling	Pooling Size: (3,3), Strides: (2,2)
Inception Block 7	Reducing Channels: 3x3-48, 5x5-16; Output Channels: 1x1-32, 3x3-64, 5x5-16, Max-16
Inception Block 8	Reducing Channels: 3x3-48, 5x5-16; Output Channels: 1x1-32, 3x3-64, 5x5-16, Max-16
Global Average Pooling	-
Dense	Output Units: Num_Classes
Softmax	-

TABLE 5.1: Modified GoogLeNet architecture implemented in JAX

The other important parameters used are:

1. **Activation function** - ReLu and Leaky ReLu were tested, both gave similar performance.

2. **Optimizer choice** - Adam

3. **Batch size** - 512 samples, smaller batch sizes result in getting stuck in a local minima.

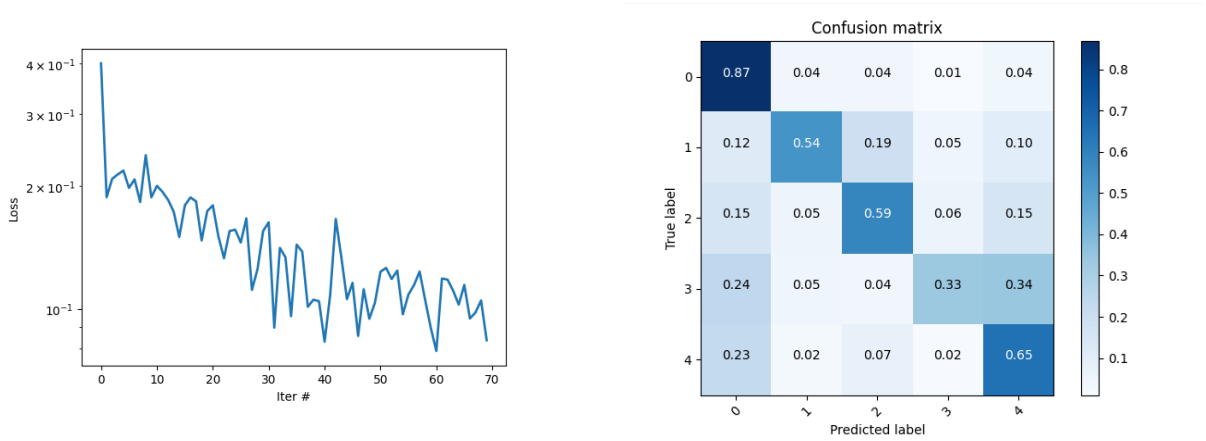
4. **Learning rate** - 10^{-3} with a decay rate of 0.9 and 1000 transition steps

5. **Loss function** - Multiclass Cross entropy loss

6. **Parameter initialization** - Kaiming normal initialization

5.1.2 Results

The original vectors were of size 1000X12, however loading the full size feature vectors were causing colab to crash during training due to memory issues. So we had to subsample the feature vectors to 300 X 12 and 500 X 12 in two different studies. When trained with 300 X 12 samples and for 2000 iterations, the google net gave an accuracy of 66.67 % whereas when trained with 500 X 12 samples 7000 iterations , the google net gave an accuracy for 69.5 %



(A) Loss Log Curve

(B) Confusion Matrix

FIGURE 5.2: Log loss curve and Confusion Matrix for 7000 iterations and feature vector size of 500 X 12

5.1.3 Comments

1. From the confusion matrix, we notice that predictions from class 3 (the hypertrophy class) is very poor. This is expected since the dataset has far fewer number of Hypertrophy classes as compared to the other classes. So we feel that the classifier hasn't seen enough of these hypertrophy samples within the 7000 iterations we have done to learn the pattern for this class accurately.

2. It is important to note that the loss function is still showing a downward trend but we had to terminate training due to colab's gpu usage limitations. We believe that training for longer times will improve accuracy.

3. It may also be useful to increase the batch size from 512 to higher values since a batch size of 128 was causing the optimizer to be stuck in local minima.

5.2 ResNet

5.2.1 Architecture and hyperparameter details

The following figure represents the ResNet architecture implemented in Jax. The concept of each residual block was described in the previous chapter.

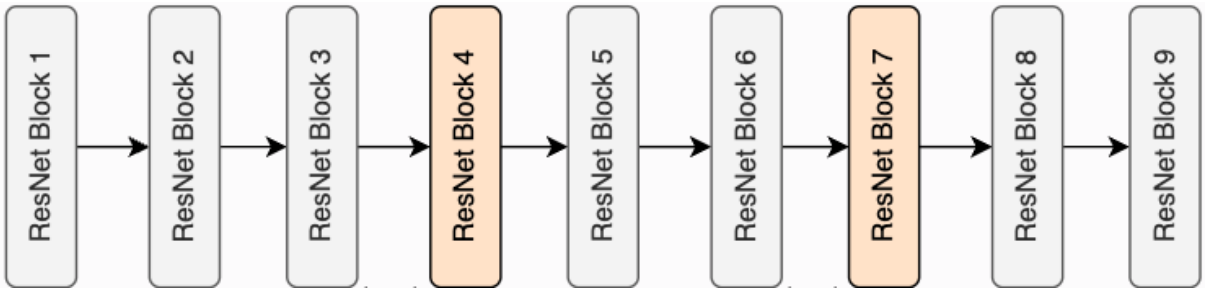


FIGURE 5.3: ResNet architecture implemented in Jax. Taken from [2]

Layer Name	Details
Convolution 1	Output channels - 16, Kernel size - (3,3)
BatchNorm 1	NA
ReLU 1	NA
ResNet Block 1	Output channels - 16, Kernel size - (3,3)
ResNet Block 2	Output channels - 16, Kernel size - (3,3)
ResNet Block 3	Output channels - 16, Kernel size - (3,3)
ResNet Block 4	Output channels - 32, Kernel size - (2,2)
ResNet Block 5	Output channels - 32, Kernel size - (3,3)
ResNet Block 6	Output channels - 32, Kernel size - (3,3)
ResNet Block 7	Output channels - 64, Kernel size - (2,2)
ResNet Block 8	Output channels - 64, Kernel size - (3,3)
ResNet Block 9	Output channels - 64, Kernel size - (3,3)
Global Average Pooling	NA
Dense	Output - 5 X 1 vector

TABLE 5.2: ResNet architecture details

The other important parameters used are:

1. **Activation function** - ReLu and Leaky ReLu were tested, both gave similar performance.

2. **Optimizer choice** - Adam optimizer

3. **Batch size** - 512 samples, smaller batch sizes result in getting stuck in a local minima.

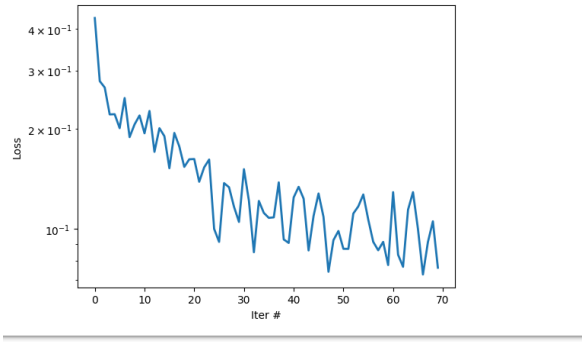
4. **Learning rate** - 10^{-3} with a decay rate of 0.9 and 1000 transition steps

5. **Loss function** - Multiclass Cross entropy loss

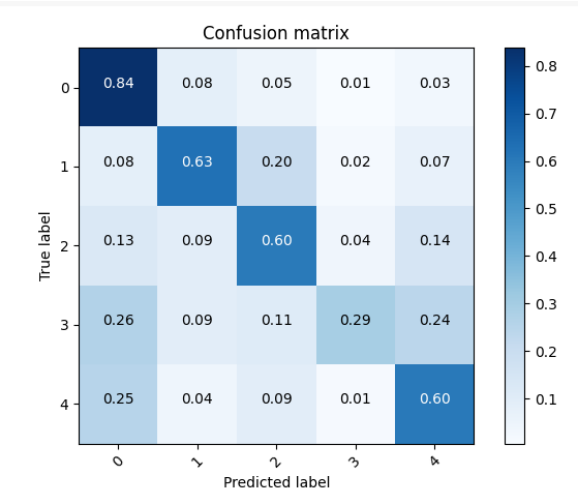
6. **Parameter initialization** - Variance scaling initialization, mode is 'fan-out', distribution is 'normal'

5.2.2 Results

The original vectors were of size 1000 X 12, however loading the full size feature vectors were causing colab to crash during training due to memory issues. So we had to subsample the feature vectors to 300 X 12 and 500 X 12 in two different studies. The accuracy for ResNet with 500 samples and 7000 iterations is 69.369. The F1 score is 0.69.



(A) Loss Log Curve



(B) Confusion Matrix

FIGURE 5.4: Log loss curve and Confusion Matrix for 7000 iterations and feature vector size of 500

5.2.3 Comments

1. The loss function seems to be exhibits a more stable downwards trend as compared to GoogLeNet. This is expected since the residual connections will lead to more stable gradient propagation. It is important to note that the loss function is still showing a downward trend but we had to terminate training due to colab's gpu usage limitations. We believe that training for longer times will improve accuracy.

2. Similar to GoogLeNet, the confusion matrix show that predictions from class 3 (the hypertrophy class) is very poor. This is expected since the dataset has far fewer number of Hypertrophy classes as compared to the other classes. So we feel that the classifier hasn't

seen enough of these hypertrophy samples within the 7000 iterations we have done to learn the pattern for this class accurately.

3. It may also be useful to increase the batch size from 512 to higher values since a batch size of 128 was causing the optimizer to be stuck in local minima.

5.3 Google Net with skip connections

5.3.1 Details

After noticing the benefits of skip connections in propogating more stable gradients, we decided to modify the original GoogLeNet arcitecture and incorporate skip connections between blocks to observe the results. The following figure represents the architecture implemented in Jax. 'IB' refers to inception block which was described previously. There are batch normalization modules within each inception block as well.

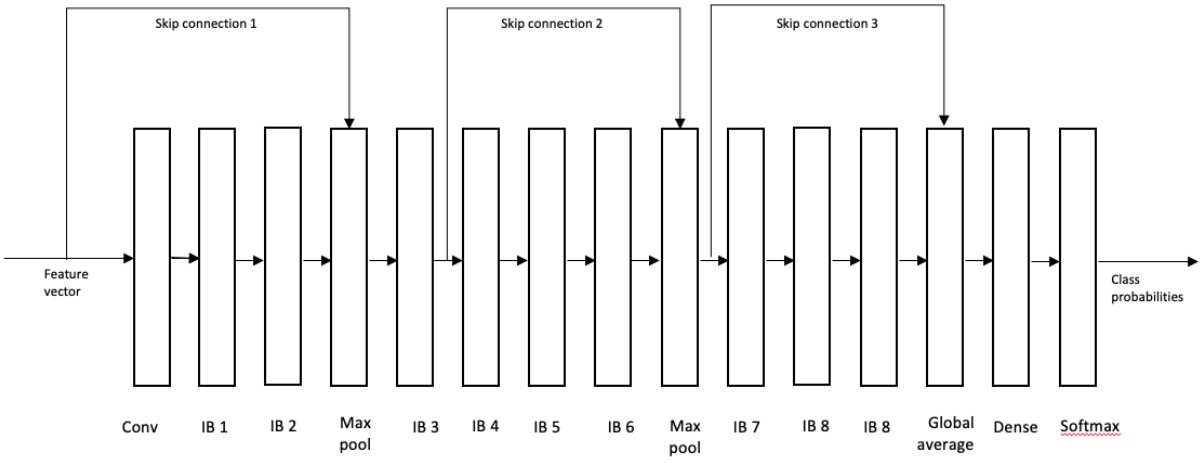


FIGURE 5.5: One variant of GoogLeNet with skip connections implemented. 'IB' refers to Inception Block described in previous chapters

The following table is a breakdown of all the parameters used for each block in the architecture.

Layer Type	**Details**
Convolution	Kernel: (3,3), No. of Filters: 96, Activation: ReLU
Batch Normalization	Momentum: 0.9, Epsilon: 0.001
Inception Block 1	Reducing Channels: 3x3-32, 5x5-16 ;Output Channels: 1x1-16, 3x3-32, 5x5-8, Max-8
Inception Block 2	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-24, 3x3-48, 5x5-12, Max-12
Max Pooling	Pooling Size: (3,3), Strides: (2,2)
Inception Block 3	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-24, 3x3-48, 5x5-12, Max-12
Inception Block 4	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-16, 3x3-48, 5x5-16, Max-16
Inception Block 5	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-16, 3x3-48, 5x5-16, Max-16
Inception Block 6	Reducing Channels: 3x3-32, 5x5-16; Output Channels: 1x1-24, 3x3-48, 5x5-12, Max-12
Max Pooling	Pooling Size: (3,3), Strides: (2,2)
Inception Block 7	Reducing Channels: 3x3-48, 5x5-16; Output Channels: 1x1-32, 3x3-64, 5x5-16, Max-16
Inception Block 8	Reducing Channels: 3x3-48, 5x5-16; Output Channels: 1x1-32, 3x3-64, 5x5-16, Max-16
Global Average Pooling	-
Dense	Output Units: Num_Classes
Softmax	-

TABLE 5.3: Modified GoogLeNet with skip connections architecture implemented in JAX

The other important parameters used are:

1. **Activation function** - ReLu and Leaky ReLu were tested, both gave similar performance.

2. **Optimizer choice** - Adam

3. **Batch size** - 512 samples, smaller batch sizes result in getting stuck in a local minima.

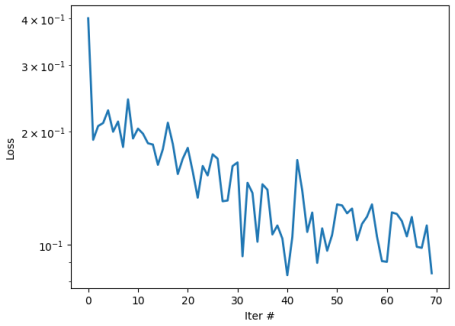
4. **Learning rate** - 10^{-3} with a decay rate of 0.9 and 1000 transition steps

5. **Loss function** - Multiclass Cross entropy loss

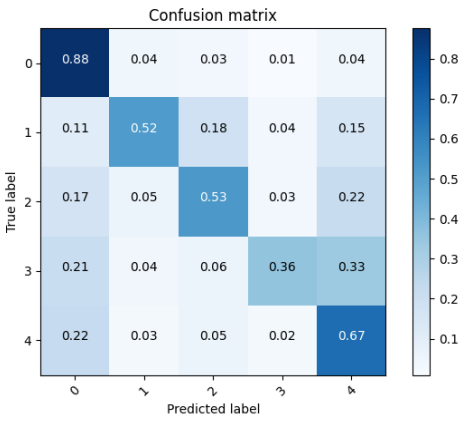
6. **Parameter initialization** - Kaiming normal initialization

5.3.2 Results

The original vectors were of size 1000 X 12, however loading the full size feature vectors were causing colab to crash during training due to memory issues. So we had to subsample the feature vectors to 300 X 12 and 500 X 12 in two different studies.The accuracy for Google net with skip connections for 500 samples and 7000 iterations was 68.6 %.



(A) Loss Log Curve



(B) Confusion Matrix

FIGURE 5.6: Log loss curve and Confusion Matrix for for GoogLeNet with skip connections 7000 iterations and feature vector size of 500

5.3.3 Comments

1. We tried out multiple variants of skip connections between the layers while adjusting the number of filters so as to ensure no broadcasting errors occur. All variants of skip connections between layers that we tried had similar performance. With more training time, it might be possible to figure out which variant of skip connection GoogLeNet may be better, but given the limited training time colab provides we could not see a clear difference.

2. The loss function seems to be exhibits a more stable downwards trend as compared to normal GoogLeNet. This is expected since the residual connections will lead to more stable gradient propagation. It is important to note that the loss function is still showing

a downward trend but we had to terminate training due to colab's gpu usage limitations. We believe that training for longer times will improve accuracy.

3. Similar to GoogLeNet, the confusion matrix show that predictions from class 3 (the hypertrophy class) is very poor. This is expected since the dataset has far fewer number of Hypertrophy classes as compared to the other classes. So we feel that the classifier hasn't seen enough of these hypertrophy samples within the 7000 iterations we have done to learn the pattern for this class accurately.

4. It may also be useful to increase the batch size from 512 to higher values since a batch size of 128 was causing the optimizer to be stuck in local minima.

5.4 LSTMs

We attempted to implement LSTMs couipled with a dense layer to create a classifier using the template Flax code provided in class. However, we ran into an issue while creating lags for the time series. Even when we were considering training data of size 18000X 500 X 12, we were running out of memory on colab when we tried to create 10 lags out of this time series. The output vector should be 18000 X 500 X 10 X 12. We believe that we can fix this issue given more time and computing resources. This isa why LSTMs were abandoned in favour of more variants of CNNs.

Chapter 6

Conclusions and Future works

In this project, we have explored the use of machine learning and deep learning techniques to classify ECG recordings into their respective diagnostic categories. We have implemented existing CNN architectures such as ResNet and GoogLeNet in Jax using the Flax module. We also attempted to modify the traditional GoogLeNet by incorporating skip connections at different frequencies between blocks in the hopes of propogating more stable gradients during training. We feel that we have succeeded to a large extent. All three architectures we implemented give reasonable accuracy on the testing dataset with GoogLeNet outperforming ResNet by a small margin. However, the loss curves indicate that if trained for longer periods of time, ResNet and/or GoogLeNet with skip connections will lead to higher accuracies.

The biggest limitation we faced is the computing power. The training times are very large for such deep models and the very high dimensional feature vectors were causing Colab to run out of memory, forcing us to subsample our feature vectors.

In terms of future works, we feel the CNN architectures we have implemented can give far better accuracy by simply extending the training time and/or using the full dimensional feature vector. We also plan to implement LSTMs to compare it's performance to CNNs. Another avenue of

interest would be to downsize the feature vectors using PCA before pushing it through the Neural Networks. This may help in reducing the training time and number of parameters.

In terms of the larger scientific and Medical community, the development of accurate and efficient algorithms for ECG classification is crucial in improving patient outcomes in healthcare. However, there are still limitations and ethical considerations that need to be addressed when developing and implementing these algorithms in healthcare settings. The lack of standardized evaluation metrics and potential biases in the data used for training and testing models are important considerations for future research.

Moving forward, future research directions should focus on addressing these limitations and evaluating the performance of these algorithms in real-world clinical settings. Additionally, exploring the use of explainable AI methods to increase transparency and interpretability of these models could aid in their adoption by healthcare professionals.

Overall, the development of accurate and efficient ECG classification algorithms has the potential to greatly benefit the medical community, and continued research in this field is essential.

Chapter 7

References

1. *"PTB-XL, a large publicly available electrocardiography dataset"* , P. Wagner et.al, Nature, Scientific Data (2020)

2. UvA deep learning online course website (basic codes for GoogLeNet and ResNet which were modified as per requirement)

3. *"Deep Learning for ECG Analysis: Benchmarks and Insights from PTB-XL"*, N. Strodthoff et.al, IEEE Journal of Biomedical and Health Informatics (May 2021)

4. *"European Cardiovascular disease statistics, Belgium; European Heart network"*(2017)

5. *"National ambulatory med. care survey : 2016 National summary tabels"*, CDC tech report (2019)

6. *"Competency in interpretation of 12 lead electrocardiograms: A summary and appraisal of published evidence"* , Ann. Internal Med, Vol 138, no 9, p. 751, May 2003