Automated Cardiac Disease Challenge (ACDC)

Eliza Giane, Shirui Li, Lydia Yang

I. Introduction

This project proposes a solution to the ACDC Challenge, which can be found at the link here. The ACDC Challenge consists of real anonymized and regulated clinical exams from the University of Dijon, and is defined as covering several well-defined pathologies with enough cases to properly train machine learning methods and clearly assess the variations of the main physiological parameters obtained from cine-MRI.

II. Previous Solutions

In the past, researchers have used a 1D CNN architecture to try to predict heart disease, which is the same problem that we are working on (Hussain 2021). By using a 1D CNN rather than a conventional 2D CNN, the convolution operation is only applied to one dimension, so the architecture is shallow and can be trained on just a normal CPU. With this architecture, it is helpful to find useful hierarchical features from the dataset to use in classification. Using this architecture, they were able to achieve an accuracy of 97% on the training set and 96% on the test set.

III. Dataset

The dataset is composed of 150 exams from all different patients and is divided into 5 evenly distributed subgroups, containing 4 pathological and 1 healthy subject group. Each patient has several channels of images, each of which has several image slices, and the number of channels and slices vary across patients. Our goal is to develop a model that will be able to accurately predict which of the five subgroups an image slice belongs to.

IV. Proposed Methods

Inspired by the research described in the Hussain 2021 paper, we also opted for a 1D Convolutional Neural Network model. We hope that with a 1D model, we can make it suitable for real-time fault detection and monitoring, which is important when classifying heart diseases in the field. We include the following proposed model:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_2 (Conv1D)           (None, 107, 9)            66537

 max_pooling1d_2 (MaxPoolin  (None, 106, 9)            0
 g1D)

 dropout_3 (Dropout)         (None, 106, 9)            0

 conv1d_3 (Conv1D)           (None, 59, 9)             3897

 max_pooling1d_3 (MaxPoolin  (None, 29, 9)             0
 g1D)

 dropout_4 (Dropout)         (None, 29, 9)             0

 flatten_1 (Flatten)         (None, 261)               0

 dense_2 (Dense)             (None, 128)               33536

 dropout_5 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 104615 (408.65 KB)
Trainable params: 104615 (408.65 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

V. Evaluation Methods

When preprocessing the data, we designated 20% of the dataset to be used as testing data after training the model. We call the testing data as x_test and y_test. We evaluate the model by calling the function `model.evaluate(x_test,y_test)`, in which we monitor test loss and test accuracy. This gives us a quantitative measure of how well the model performs on unseen data, which ensures that the model hasn't overfit to the training data and that it can generalize well to new, unseen examples.

We also use the function `model.predict(x_test)` to analyze the predictions through a classification report, calculated balance accuracy score, and a confusion matrix.

The classification report provides us with key classification metrics such as precision, recall, and F1-score for each class in the dataset. It gives a more detailed understanding of how well the model performs for each class, which allows us to later investigate why the model performs better with certain classes but not the others.

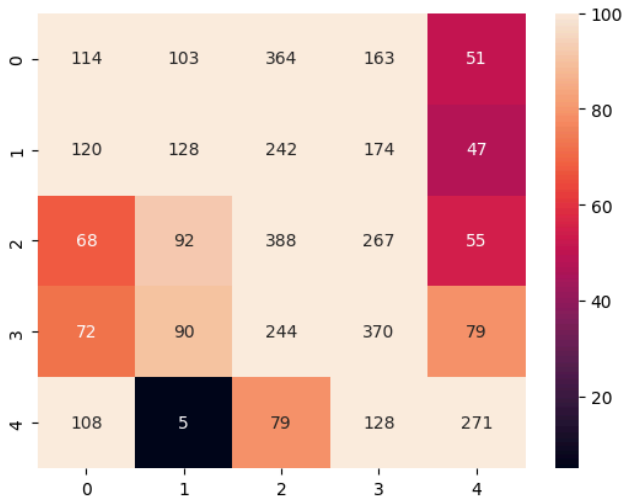
We opted for calculating the balanced accuracy score because it can provide a more accurate measure of the overall model performance.

Finally, the confusion matrix allows us to visualize the performance of the model's classification algorithm, helping us better understand where it is making mistakes.

By combining model.evaluate() and model.predict(), we were able to get a more comprehensive understanding of the model's performance, both in terms of overall metrics and in more nuanced analysis.

VI. Results

We have obtained 33% test accuracy as well as 86% test and validation accuracy. Given that these results are not very good, we would hope to improve them by improving our model. The classification report showed that precision, recall, and f1-score hovered around 32-34%, which means that the precision and recall were similar values, which they are. The balanced accuracy score was 33%, so this accounts for the balance of the data. Lastly, the confusion matrix is shown below:



The confusion matrix shows that there is some right classification, but there is a lot to be worked on. The diagonal is not super dark, and the values around the diagonal are relatively high.

VII. Discussion

Given that the accuracy is not very good as it currently is, we would hope to improve them by improving our model. Instead of creating our own model, we hope to use a pre-trained model already to give us a stepping stone on which to build a better model. In addition, there is a large disparity between the test accuracy and the training and validation accuracy, which may mean that there is overfitting going on. Due to time constraints, we could not load the entire dataset and we only took a portion of it to train on. If we were able to load the entire dataset, that could solve our overfitting problem. Another solution would be to make the model less complex and therefore less likely to overfit.