Cardiomegaly Disease Classification Using Convolutional Neural Networks

Altaaf Ally (2424551)
Hamzah Mia (2430188)
Rayhaan Hanslod (2430979)
Hamdullah Dadabhoy (2441030)

May 20, 2024

Table of Contents

1 Introduction

Cardiomegaly, an enlargement of the heart, is often a sign of underlying cardiovascular conditions. Early detection is crucial for effective medical intervention. In this study, we propose a CNN-based model for predicting cardiomegaly from chest X-ray images.

2 Dataset

The dataset used in this study was sourced from Kaggle, a popular platform for hosting datasets and machine learning competitions. It contains chest X-ray images divided into two folders: "test" and "train". Each folder further contains two subfolders labeled "yes" and "no", indicating the presence or absence of a disease in the corresponding X-ray images.

2.1 Dataset Description

The dataset is a processed version of the original NIH Chest X-ray Dataset. Using the CSV information provided, the author separated images of Cardiomegaly and applied contrast limited adaptive histogram equalization (CLAHE) to enhance the image quality. The images were then resized to a uniform dimension of 128x128 pixels. The training and testing images are equally divided in a 1:1 ratio.

2.2 Acknowledgements

The author expresses gratitude to the NIH Clinical Center for providing one of the largest publicly available chest X-ray datasets to the scientific community.

2.3 Dataset Objectives

The objective of this study is to develop a machine learning model capable of accurately classifying chest X-ray images as positive or negative for cardiomegaly using the provided dataset. By leveraging the labeled images, we aim to create a robust model that can assist in the automated detection of cardiomegaly from chest X-ray images.

3 Preprocessing

The dataset underwent several preprocessing steps to prepare the images for training the machine learning model. These steps were performed using the `ImageDataGenerator` class from the Keras library.

3.1 Resizing

All images in the dataset were resized to a uniform target size of 128×128 pixels. This ensures that the input dimensions of the images are consistent across the entire dataset, which is essential for training the model. The resizing was performed using the `target_size` parameter in the `flow_from_directory` method of the `ImageDataGenerator`.

3.2 Grayscale Conversion

The images were converted from their original color space to grayscale. This simplifies the image data and reduces the computational complexity of the model. Grayscale images contain only one channel, representing the intensity of each pixel. The conversion to grayscale was achieved by setting the `color_mode` parameter to `'grayscale'` in the `flow_from_directory` method.

3.3 Normalization

Pixel values in the images were normalized to the range [0, 1]. Normalization is a common preprocessing step that helps to standardize the input data and improve the convergence of the model during training. The normalization was performed by setting the `rescale` parameter to `1./255` in the `ImageDataGenerator` constructor. This rescales each pixel value by dividing it by 255, effectively converting the pixel values from the range [0, 255] to [0, 1].

3.4 Data Splitting

The training data was split into training and validation subsets using the `validation_split` parameter in the `ImageDataGenerator` constructor. In this setup, 20% of the training data was allocated for validation, while the remaining 80% was used for training. Additionally, a separate test folder containing all the pictures designated for testing was used to evaluate the final model performance. The validation subset is used to monitor the model's performance during training, aiding in hyperparameter tuning and preventing overfitting. The test set provides an unbiased evaluation of the model's effectiveness after the training process is complete.

3.5 Batch Generation

The preprocessed data was generated in batches using the `flow_from_directory` method of the `ImageDataGenerator`. This method automatically reads images from the specified directory, applies the preprocessing steps, and yields batches of data ready for training. The `batch_size` parameter was set to 32, indicating that each batch contains 32 images. The `class_mode` parameter was set to `'binary'` for binary classification, as the images are labeled as either positive or negative for cardiomegaly.

The preprocessing steps were applied consistently across the training, validation, and test sets. For the test set, only the normalization step was applied using a separate `ImageDataGenerator` instance, as resizing and grayscale conversion were already performed during the dataset creation. The `shuffle` parameter was set to `False` for the test generator to ensure that the order of the test images remains unchanged.

These preprocessing steps – resizing, grayscale conversion, normalization, data splitting, and batch generation – prepare

the dataset for training and evaluating the machine learning model. They help to standardize the input data, reduce computational complexity, and facilitate the efficient training process.

4 Model Architecture

Our proposed CNN model architecture aims to effectively learn and extract relevant features from chest X-ray images to predict the presence of cardiomegaly. The model consists of a sequence of convolutional layers, pooling layers, and fully connected layers designed to progressively extract hierarchical representations from the input images.

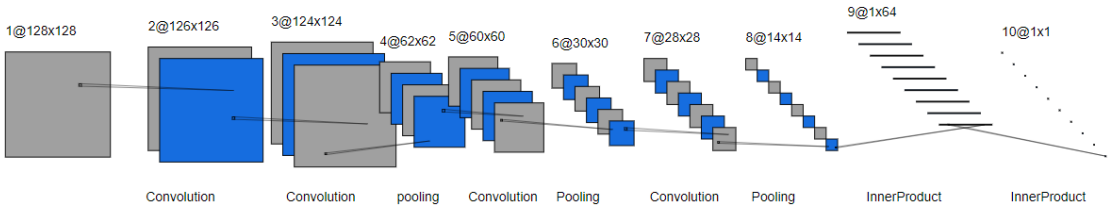

Figure 1: Architecture of the proposed CNN model for cardiomegaly prediction.

The input layer of the model accepts grayscale images of size 128x128 pixels. These images undergo processing through three sets of convolutional and pooling layers. The first convolutional layer applies 32 filters of size 3x3 to capture basic features like edges and textures. Subsequently, a pooling layer reduces the spatial dimensions to 124x124 pixels, enhancing computational efficiency and focusing on essential features.

The second convolutional layer employs 64 filters, enabling the model to learn more complex and abstract features from the input images. Another pooling layer further reduces the spatial dimensions to 60x60 pixels. The third convolutional layer extends the model's capacity with 128 filters, capturing high-level representations of features. A final pooling layer reduces dimensions to 28x28 pixels.

After convolutional and pooling layers, the output is flattened into a 1D vector, ready for fully connected layers. The first fully connected layer, consisting of 64 units with ReLU activation, learns high-level representations of extracted features. The second fully connected layer produces the final output, representing the probability of cardiomegaly presence, with a sigmoid activation function ensuring values between 0 and 1.

Our CNN architecture balances model complexity and computational efficiency, allowing it to learn hierarchical features from input images. By carefully designing the arrangement and configuration of layers, our model efficiently captures both local and global patterns, essential for accurate cardiomegaly prediction.

Figure 1 visually represents our model architecture, illustrating the flow of data through different layers and gradual reduction of spatial dimensions. This architecture aims to optimize performance by effectively learning and extracting relevant features from chest X-ray images for reliable cardiomegaly prediction.

5 Experimental Setup and Training

To evaluate the performance of our CNN model for cardiomegaly prediction, we conducted a series of experiments using the preprocessed chest X-ray dataset. The experiments aimed to investigate the impact of different hyperparameters and loss functions on the model's performance.

5.1 Training Setup

The CNN model was implemented using the Keras library with TensorFlow as the backend. The model architecture consists of convolutional layers, pooling layers, and fully connected layers. The input to the model is a grayscale image of size 128x128 pixels.

During training, the dataset was split into training, validation, and test sets. The training set was used to optimize the model's parameters, while the validation set was used to monitor the model's performance and prevent overfitting. The test set was used to evaluate the final performance of the trained model.

Data augmentation techniques were applied to the training set to increase the diversity of the samples and improve the model's generalization ability. The augmentation techniques included random rotations, shifts, and flips of the input images.

5.2 Hyperparameter Tuning

To find the optimal hyperparameters for the CNN model, we performed a grid search over a range of values. The primary hyperparameter of interest was the learning rate.

The learning rate determines the step size at which the model's weights are updated during training. We explored a range of learning rates, including 0.001, 0.01, and 0.1, to find the value that resulted in the best performance.

The batch size, which determines the number of samples processed in each iteration during training, was kept constant throughout the experiments to ensure consistency and comparability of results.

The grid search was performed using a validation split from the training data. The `validation_split` parameter was used to allocate 20% of the training data for validation purposes. The model's performance on this validation subset was used to select the best hyperparameters.

5.3 Loss Functions

To investigate the impact of different loss functions on the model's performance, we trained the CNN model with three commonly used loss functions for binary classification tasks:

1. Binary Cross-Entropy: This loss function measures the disparity between the predicted probabilities and the true labels. It is widely used for binary classification problems.

2. Hinge Loss: This loss function is used for maximum-margin classification. It encourages the model to make confident predictions by penalizing predictions that are close to the decision boundary.

3. Mean Squared Error: This loss function measures the average squared difference between the predicted probabilities and the true labels. It is commonly used for regression tasks but can also be used for classification.

The model was trained separately with each loss function, keeping the other hyperparameters fixed. The performance of the model with each loss function was evaluated using the test set.

5.4 Training Process

The CNN model was trained using the Adam optimizer, which adapts the learning rate for each parameter based on its historical gradients. The model was trained for a maximum of 50 epochs, with early stopping employed to prevent overfitting. Early stopping monitored the validation loss and stopped the training if no improvement was observed for a specified number of epochs.

During training, the model's weights were saved at each epoch using model checkpointing. This allowed us to retrieve the best-performing model based on the validation loss.

The training process was repeated for each combination of hyperparameters and loss functions.

6 Model Evaluation

To evaluate the performance of our trained CNN model, we conducted a comprehensive analysis using various metrics and visualizations. The evaluation process aimed to assess the model's ability to accurately predict cardiomegaly in chest X-ray images.

6.1 Training and Validation Accuracy Analysis

Figure 2 illustrates the training and validation accuracy for each loss function used in our experiments. The key observations are as follows:

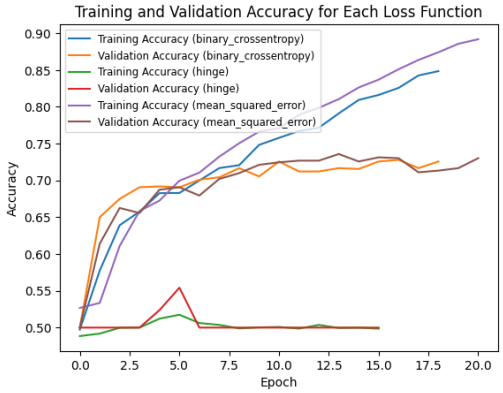

Figure 2: Training and validation accuracy for each loss function.

6.1.1 Binary Crossentropy

Training Accuracy: The model trained with the binary crossentropy loss function shows a steady increase in training accuracy, reaching close to 0.85 by the end of the epochs.

Validation Accuracy: The validation accuracy for this loss function also improves over time, though with some fluctuations, indicating that the model performs consistently well on unseen data but has some variance.

6.1.2 Hinge

Training Accuracy: The model using the hinge loss function had a constant training accuracy around 0.5, with fluctuations but never exceeding 0.55.

Validation Accuracy: However, the validation accuracy is relatively low and shows significant fluctuations, which suggests that the model may be struggling to generalize well when trained with this loss function with a max accuracy of 0.55.

6.1.3 Mean Squared Error

Training Accuracy: For the mean squared error loss function, the training accuracy shows a rapid increase initially, surpassing 0.8 and reaching a maximum of 0.9.

Validation Accuracy: The validation accuracy follows a similar trend to the training accuracy but remains slightly lower ranging between 0.7 to 0.75 as the number of epochs increased. This indicates that while the model learns quickly, it may be prone to overfitting as the training and validation curves diverge.

6.1.4 Conclusion

Best Performing Loss Function: The binary crossentropy and mean squared error loss function appears to be the most effective for our model, achieving the highest training and validation accuracies with relatively stable performance.

Challenges with Hinge Loss: The hinge loss function does not perform as well, particularly in terms of validation and training accuracy, suggesting that it may not be suitable for this specific problem.

7 Evaluation of Training Losses for Different Loss Functions

The graph in Figure 3 illustrates the training losses for three different loss functions: binary cross-entropy, hinge, and mean squared error. By analyzing the convergence rates, initial and final loss values, and stability of each loss function, we can gain insights into their performance during the training process.

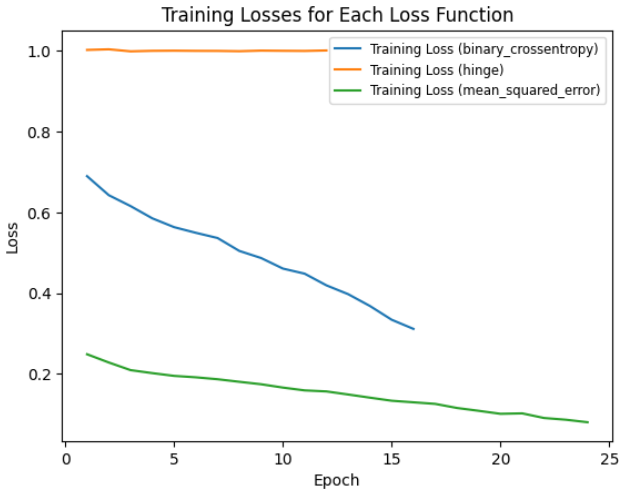

Figure 3: Training losses for different loss functions over 25 epochs

7.1 Convergence Rates

The binary cross-entropy loss exhibits the fastest convergence rate among the three loss functions. It starts with a high loss value but decreases rapidly in the early epochs, indicating quick learning and improvement in the model's performance. The hinge loss remains consistent throughout the training process, neither increasing nor decreasing, suggesting a stable but limited learning progression. The mean squared error loss has the slowest convergence rate, with a relatively smooth and gradual decrease in loss values across all epochs.

7.2 Initial Loss Values

The binary cross-entropy loss has the highest initial loss value, starting around 0.5. The hinge loss has a consistent 1.0 loss. The mean squared error loss has the lowest initial loss value, starting above 0.2.

7.3 Final Loss Values

After 25 epochs, binary cross-entropy and mean squared error loss functions converge to relatively low loss values, indicating successful training of the model. The binary cross-entropy loss final loss values, is just above 0.3. The mean squared error loss converges to the lowest final loss value, just below 0.1.

8 Evaluation of Test Accuracy for Different Loss Functions

The bar graph in Figure 4 presents the test accuracy achieved by the model using three different loss functions: binary cross-entropy, hinge, and mean squared error. By comparing the test accuracy values, we can assess the model's performance on unseen data and determine the effectiveness of each loss function.

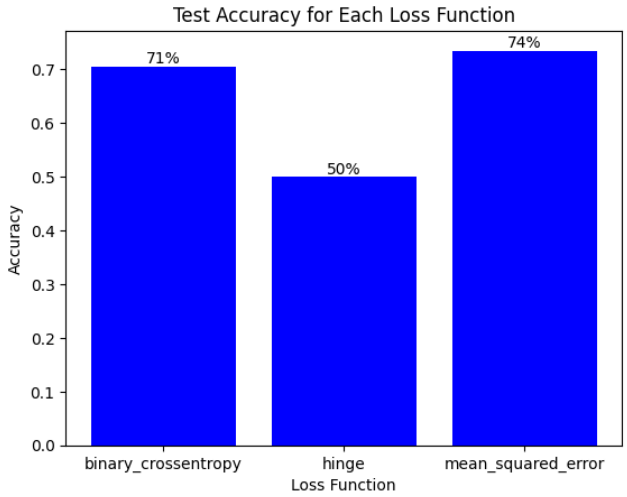

Figure 4: Test accuracy for different loss functions

8.1 Test Accuracy Values

The mean squared error loss function achieves the highest test accuracy of 74%, outperforming the other two loss functions. The binary cross-entropy loss function comes in second with a test accuracy of 71%, while the hinge loss function yields the lowest test accuracy of 50

8.2 Comparison of Loss Functions

The difference in test accuracy values highlights the impact of the choice of loss function on the model's performance. The mean squared error loss function demonstrates superior performance on the test data, indicating its effectiveness in capturing the underlying patterns and generalizing well to unseen examples. Although mean squared error is commonly used for regression tasks, it shows promising results for this binary classification problem. The binary cross-entropy loss function, which is widely used for binary classification, also achieves a relatively high test accuracy of 71%. However, the hinge loss function performs poorly with a test accuracy of 50%, suggesting that it may not be suitable for this particular classification task.

9 Evaluation of Test Accuracy for Different Learning Rates

The bar graph in Figure 5 presents the test accuracy achieved by the model using binary cross-entropy loss function with three different learning rates: 0.001, 0.01, and 0.1. By comparing the test accuracy values, we can assess the model's performance on unseen data and determine the effectiveness of each learning rate.

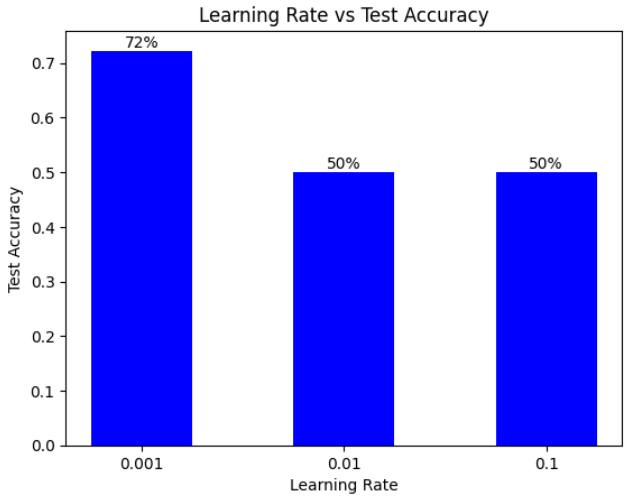

Figure 5: Test accuracy for different learning rates

9.1 Test Accuracy Values

- **Learning Rate 0.001:**
  - Test Accuracy: 72%
  - This learning rate achieves the highest test accuracy among the three rates tested. A learning rate of 0.001 allows the model to learn effectively from the data, leading to better generalization and higher accuracy on the test set.
- **Learning Rate 0.01:**
  - Test Accuracy: 50%
  - This learning rate results in a significantly lower test accuracy compared to 0.001. The accuracy is 50%, indicating that the model might not be learning as effectively, possibly due to the learning rate being too high, which can cause the model to overshoot optimal parameters.
- **Learning Rate 0.1:**
  - Test Accuracy: 50%
  - Similar to the learning rate of 0.01, a learning rate of 0.1 also achieves a test accuracy of 50%. This suggests that the learning rate is too high, leading to poor performance, likely due to the model's inability to converge properly.

9.2 Summary

A lower learning rate of 0.001 results in the highest test accuracy (72%), indicating effective learning and good generalization. Higher learning rates of 0.01 and 0.1 both result in lower test accuracy (50%), suggesting that these rates are too high and may cause convergence issues or poor learning dynamics. For this particular model and dataset, a learning rate of 0.001 is the most effective choice for achieving high test accuracy.

10 Confusion Matrix Analysis

To further evaluate the performance of the trained model, we generate a confusion matrix using the binary cross-entropy loss function. The confusion matrix provides a detailed breakdown of the model's predictions, allowing us to assess its ability to correctly classify the samples into their respective classes. By examining the values in the confusion matrix, we can gain insights into the model's strengths and weaknesses, as well as identify any potential areas for improvement.

10.1 Confusion Matrix Values

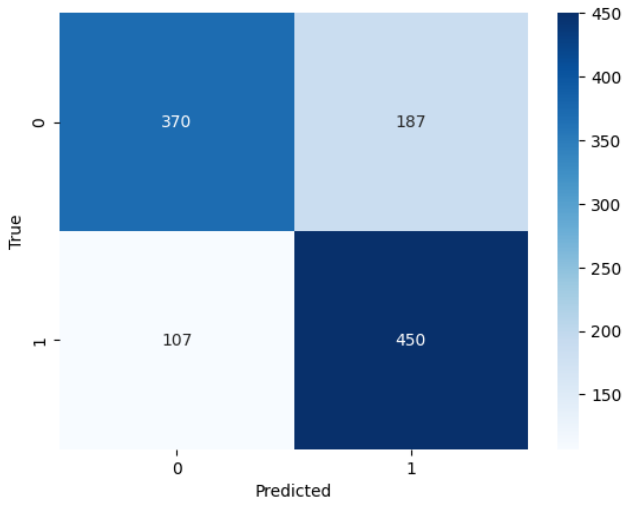

Figure 6: Confusion Matrix

The confusion matrix, shown in Figure 6, displays the following values:

- True Negatives (TN): 370
- False Positives (FP): 187
- False Negatives (FN): 107
- True Positives (TP): 450

10.2 Interpretation

- **True Negatives (370):** The model correctly predicted 370 instances as negative (class 0).
- **False Positives (187):** The model incorrectly predicted 187 instances as positive (class 1) that are actually negative.
- **False Negatives (107):** The model incorrectly predicted 107 instances as negative (class 0) that are actually positive.
- **True Positives (450):** The model correctly predicted 450 instances as positive (class 1).

10.3 Performance Metrics

From the confusion matrix, we can derive several performance metrics:

- **Accuracy:** $\frac{TP+TN}{TP+TN+FP+FN} = \frac{450+370}{450+370+187+107} \approx 0.79$
- **Precision:** $\frac{TP}{TP+FP} = \frac{450}{450+187} \approx 0.71$
- **Recall:** $\frac{TP}{TP+FN} = \frac{450}{450+107} \approx 0.81$
- **F1 Score:** $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall} \approx 0.76$

10.4 Summary

The model shows a reasonably high accuracy of approximately 79%. The precision indicates that about 71% of the predicted positive instances are correct, while the recall indicates that the model correctly identifies about 81% of the actual positive instances. The F1 score, which balances precision and recall, is approximately 0.76, reflecting a good overall performance.

While the model performs well, there are still some misclassifications (187 false positives and 107 false negatives). Addressing these misclassifications can further improve the model's performance. Techniques such as tuning the decision threshold, using a different model, or employing more advanced techniques like ensemble learning may help reduce these errors.

11 Conclusion

In this study, we explored the impact of different loss functions on the performance of a CNN model for the binary classification of cardiomegaly using chest X-ray images. Accurate classification of this cardiovascular condition is crucial for early detection and timely medical intervention. Our experiments revealed that the mean squared error loss function achieved the highest test accuracy of 74%, outperforming commonly used alternatives. These findings highlight the importance of selecting appropriate loss functions and conducting comprehensive evaluations to optimize model performance in the context of medical image classification.

The confusion matrix analysis provided insights into the model's ability to correctly classify positive and negative samples, emphasizing the need to minimize false negatives in cardiomegaly detection. By considering problem-specific requirements and dataset characteristics, we can develop more reliable and effective classification systems to assist medical professionals in diagnosing cardiomegaly.

Looking ahead, the potential of CNN models for binary classification in the medical domain is immense. With advancements in deep learning techniques and the increasing availability of large-scale medical imaging datasets, we can develop sophisticated models for various disease classifications. These models can serve as valuable tools for screening, early detection, and decision support in clinical settings, ultimately improving patient care and outcomes.

In summary, this study underscores the importance of exploring different loss functions and conducting thorough evaluations to optimize the performance of CNN models for cardiomegaly classification. By applying the insights gained from our analysis, practitioners can make informed decisions when developing similar models, contributing to the advancement of AI-assisted diagnostic systems in the medical field.