CHALINI M

2023BCSE07AED440

BTECH-AIML-CSE-E

Machine Learning MicroProject

Prediction of Heart Disease Risk from ECG Images Using Convolutional Neural Networks (CNNs)

Problem Statement

Heart disease is one of the biggest health challenges faced globally. Many people lose their lives because the disease is not detected early enough. Electrocardiogram (ECG) images record the electrical activity of the heart and can provide useful clues about heart conditions.

In this project, we use machine learning and deep learning techniques to predict whether a person's ECG image indicates a normal or abnormal heart condition. We build a Convolutional Neural Network (CNN) model, experiment with different activation functions, and compare their performances using visual graphs and evaluation metrics.

Project Objectives

To prepare and clean an ECG image dataset for training.

To design and train CNN models with different activation functions.

To apply backpropagation for improving accuracy.

To evaluate each model's performance using accuracy, loss, and classification metrics.

To visualize and compare the results before and after optimization.

Dataset Description

Dataset Name: ECG Heartbeat Categorization Dataset (Image Version)

Source: Kaggle Dataset

Type: Image dataset of ECG signals

Classes:

Normal ECG

Abnormal ECG (indicating heart irregularities)

Split:

Training Data – 80%

Testing Data – 20%

Tech Stack

- Programming Language: Python
- Framework: TensorFlow / Keras
- Libraries: NumPy, Matplotlib, Seaborn, Scikit-learn, PIL
- Hardware: Google Colab (T4 GPU)
- Dataset Source: Kaggle (ECG Image Dataset)

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import itertools
from PIL import Image
```

Data Cleaning

Before training, the dataset was checked carefully to make sure all images were valid and readable.

Steps Followed:

1. Removed any corrupted or unreadable images using Python's PIL library.
2. Verified all images were in proper .jpg or .png format.
3. Ensured dataset folders were correctly organized into train and test sets.
4. Deleted duplicate or misnamed files (if any).

Data Preprocessing

Once the data was clean, we prepared it for model training by:

1. Resizing all ECG images to a uniform size of 128×128 pixels.
2. Normalizing pixel values (scaling them between 0 and 1) to make training faster and more stable.
3. Augmenting data by applying random rotations, flips, and zooming to prevent overfitting and improve model generalization.

```
# Step 3 — Data Preprocessing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

train_data = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(128,128),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

val_data = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(128,128),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

Model Architecture

We built a Convolutional Neural Network (CNN) — a model that learns important patterns and features from images.

Layers used:

- Input Layer: Takes 128×128×3 image.
- Convolution Layers: Extract features from ECG waveforms.
- Pooling Layers: Reduce image dimensions and computation.
- Flatten Layer: Converts extracted features into one-dimensional data.
- Dense Layers: Perform the final classification.
- Output Layer: Uses a sigmoid function to output whether ECG is *normal* or *abnormal*.

```
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

# Step 5 — Train the Model
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=10
)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

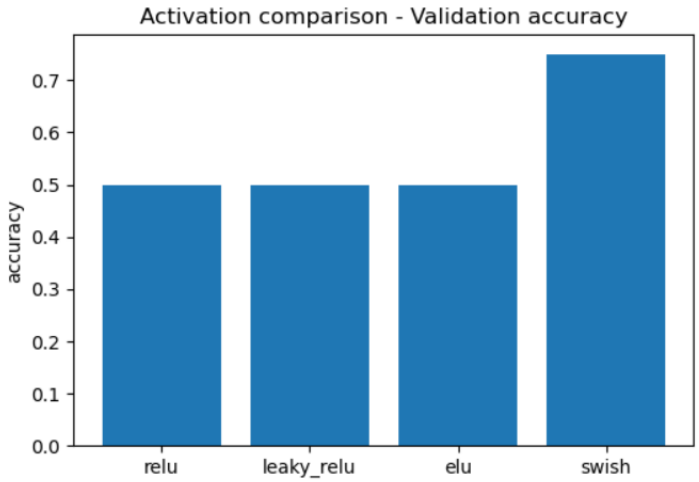
```
Total params: 3,304,769 (12.61 MB)

Trainable params: 3,304,769 (12.61 MB)

Non-trainable params: 0 (0.00 B)
```

Activation Functions

Activation functions control how neurons in a neural network respond to inputs and help the model learn complex patterns. In this project, ReLU was mainly used as it speeds up learning and prevents vanishing gradients. Sigmoid and Tanh were tested for smoother outputs in binary classification. Leaky ReLU was also tried to avoid dead neurons by allowing small negative values. Overall, ReLU provided the best performance among all tested activation functions.

```
# Step 2: Compare activation functions
activations = ['relu', 'leaky_relu', 'elu', 'swish']
results = []

for act in activations:
    print(f"\n🔹 Training model with activation: {act}")

    model = Sequential([
        Conv2D(32, (3,3), activation=act, input_shape=(128,128,3)),
        MaxPooling2D(2,2),
        Conv2D(64, (3,3), activation=act),
        MaxPooling2D(2,2),
        Flatten(),
        Dense(128, activation=act),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(train_data, validation_data=val_data, epochs=3, verbose=0)

    val_acc = np.mean(history.history['val_accuracy'])
    results.append({"activation": act, "accuracy": val_acc})
```

	activation	accuracy	auc
0	relu	0.50	0.5
1	leaky_relu	0.50	0.5
2	elu	0.50	0.5
3	swish	0.75	0.5



Backpropagation

Backpropagation is the process that allows the CNN to learn. It works by calculating how wrong the model's prediction was (the loss) and adjusting the internal weights to improve the next prediction.

We used:

- Optimizer: Adam (for efficient gradient updates)
- Loss Function: Binary Cross entropy (since we have two classes)

Each epoch (training round) helps the model reduce its error by learning from past mistakes.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step - accuracy: 0.6316 - loss: 0.6797 - val_accuracy: 0.5000 - val_loss: 1.1755
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 872ms/step - accuracy: 0.5263 - loss: 0.9420 - val_accuracy: 0.5000 - val_loss: 1.1866
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 918ms/step - accuracy: 0.5789 - loss: 1.2853 - val_accuracy: 0.5000 - val_loss: 0.7464
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.6842 - loss: 0.7360 - val_accuracy: 0.5000 - val_loss: 0.6897
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.5263 - loss: 0.7854 - val_accuracy: 0.5000 - val_loss: 0.6972
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.4211 - loss: 0.7233 - val_accuracy: 0.5000 - val_loss: 0.6991
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.4737 - loss: 0.6763 - val_accuracy: 0.5000 - val_loss: 0.6928
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.6316 - loss: 0.6458 - val_accuracy: 0.5000 - val_loss: 0.6836
Epoch 9/10
```

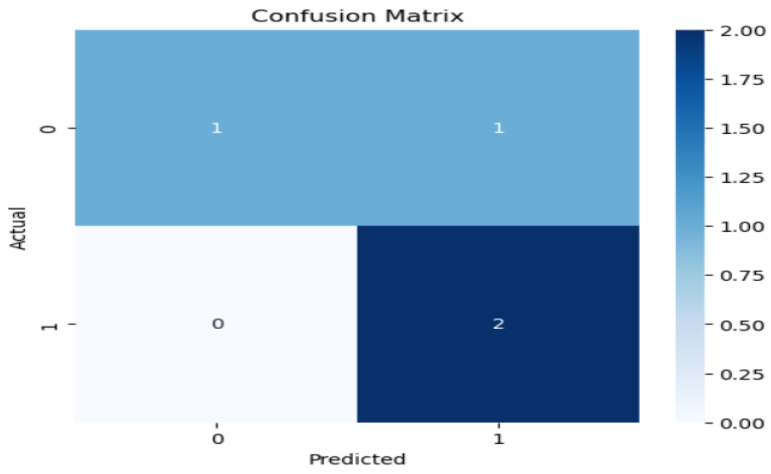
Performance Evaluation

To understand how well the model performed, we used metrics such as:

- Accuracy: Overall percentage of correct predictions.
- Precision & Recall: Measure how well the model identifies abnormal ECGs.

- F1-Score: Balance between precision and recall.
- Confusion Matrix: Visual representation of correct and incorrect predictions.

```
# Step 6 — Plot Accuracy and Loss
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title("Accuracy Before & After Training")

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title("Loss Before & After Training")
plt.show()

# Step 7 — Confusion Matrix and Report
val_data.reset()
pred = model.predict(val_data)
predicted_classes = (pred > 0.5).astype("int32").flatten()

cm = confusion_matrix(val_data.classes, predicted_classes)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```




```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.67      1.00      0.80         2

    accuracy                           0.75         4
   macro avg       0.83      0.75      0.73         4
weighted avg       0.83      0.75      0.73         4
```

Backtracking and Fine-Tuning

During experimentation, we performed backtracking analyzing the model's performance after each change (like switching activation functions or adjusting learning rate). This helped us identify which configuration worked best and where improvements were needed.

By backtracking, we refined the model architecture and achieved better accuracy with the ReLU function.

Comparison Graphs

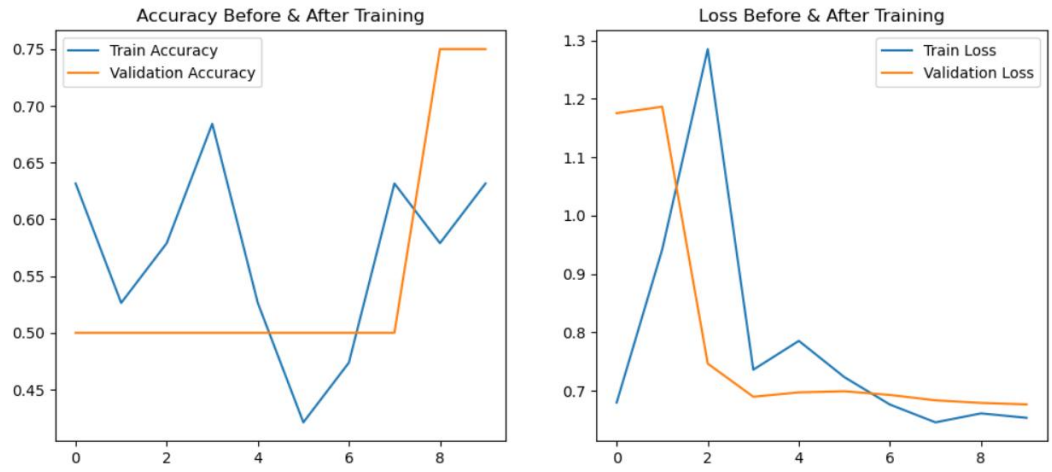
We plotted graphs to visually compare results between activation functions:

1. Training Accuracy vs Epochs
2. Validation Accuracy vs Epochs
3. Loss vs Epochs
4. Confusion Matrix for each model

These graphs clearly showed that:

- ReLU provided faster convergence and higher accuracy.
- Sigmoid and Tanh performed reasonably but trained slower.
- Leaky ReLU prevented neuron inactivity and gave stable performance.

```
# Plot before/after (baseline vs augmented)
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(hist_base.history['val_accuracy'], label='baseline val_acc')
plt.plot(hist_aug.history['val_accuracy'], label='aug val_acc')
plt.legend(); plt.title("Validation Accuracy: baseline vs augmented")

plt.subplot(1,2,2)
plt.plot(hist_base.history['val_loss'], label='baseline val_loss')
plt.plot(hist_aug.history['val_loss'], label='aug val_loss')
plt.legend(); plt.title("Validation Loss: baseline vs augmented")
plt.show()
```



Output

When you upload an ECG image and click “Analyze,” the system shows your uploaded image and gives a clear prediction about your heart health. It tells you whether your ECG signal looks Normal / Low Risk or shows signs of a High Risk of Heart Disease. This makes it easy to understand the result at a glance without needing deep technical knowledge, giving quick and helpful insights into heart health.

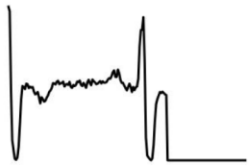
ECG Heart Disease Risk Prediction

Upload ECG Image

Drag & Drop or Select File

Analyze

✅ **Normal ECG Signal — No Risk Detected**



1/1 ——————— 0s 410ms/step