wdelsqkuj

April 26, 2025

0.1 1. Design a deep learning model using an Artificial Neural Network (ANN) to classify patients

The model architecture is:

- Input layer: Accepts all the features (age, cholesterol, blood pressure, etc.).
- Hidden Layer 1: Dense layer with 128 neurons and ReLU activation.
- Hidden Layer 2: Dense layer with 64 neurons and ReLU activation.
- Output Layer: Dense layer with 1 neuron and sigmoid activation (for binary classification).

GitHub Link: https://github.com/MrSaltyFish/aiml-assignment

0.2 2. What activation function would you use in the output layer and why?

I would use the **sigmoid activation function** in the output layer because:

- It outputs a value between 0 and 1, representing the probability that a patient has heart disease.
- Since it is a **binary classification problem** (has_disease = 0 or 1), sigmoid is the ideal choice.

0.3 3. How would you handle class imbalance if has_disease = 1 is rare?

To handle class imbalance:

- **Use class weights** while training the model. It assigns a higher penalty to misclassifying the minority class.
- Optionally, techniques like **oversampling** (SMOTE) or **undersampling** could also be used.
- Using class weights ensures that the model pays more attention to the minority class without altering the original dataset.

[49]:
```
# a. Import Libraries
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.utils import class_weight
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.optimizers import Adam

# Import libraries for evaluation
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

[50]:
```
# 1. Load Data
data = pd.read_csv('Heart.csv')
```

[51]:
```
# 2. Preprocessing
# Convert target column to binary
data['AHD'] = data['AHD'].map({'No': 0, 'Yes': 1})

# Encode categorical columns
categorical_cols = ['ChestPain', 'Thal']
data = pd.get_dummies(data, columns=categorical_cols)

# Separate features and target
X = data.drop(['AHD'], axis=1)  # Drop target and unnamed index
y = data['AHD']

# Scale numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

[52]:
```
# 3. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[53]:
```
# # 4. Calculate class weights
weights = class_weight.compute_class_weight(class_weight='balanced', classes=np.
↪unique(y_train), y=y_train)
class_weights = dict(enumerate(weights))
```

[54]:
```
# 6. Compile model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy',␣
↪metrics=['accuracy'])
# model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])
```

[55]:
```
from sklearn.utils import class_weight

# Calculate class weights for imbalanced data
class_weights = class_weight.compute_class_weight('balanced', classes=np.
↪unique(y_train), y=y_train)
class_weights = dict(enumerate(class_weights))
```

```python
# Train the model with class weights
history = model.fit(X_train, y_train, epochs=50, batch_size=16,␣
 ↪validation_split=0.2, class_weight=class_weights)
```

```
Epoch 1/50
13/13           2s 24ms/step -
accuracy: 0.4736 - loss: 0.6982 - val_accuracy: 0.4286 - val_loss: 0.6939
Epoch 2/50
13/13           0s 9ms/step -
accuracy: 0.4643 - loss: 0.6966 - val_accuracy: 0.4286 - val_loss: 0.6935
Epoch 3/50
13/13           0s 16ms/step -
accuracy: 0.4469 - loss: 0.6940 - val_accuracy: 0.4286 - val_loss: 0.6932
Epoch 4/50
13/13           0s 10ms/step -
accuracy: 0.5007 - loss: 0.6930 - val_accuracy: 0.5714 - val_loss: 0.6928
Epoch 5/50
13/13           0s 9ms/step -
accuracy: 0.5940 - loss: 0.6871 - val_accuracy: 0.5714 - val_loss: 0.6927
Epoch 6/50
13/13           0s 9ms/step -
accuracy: 0.5686 - loss: 0.6913 - val_accuracy: 0.5714 - val_loss: 0.6928
Epoch 7/50
13/13           0s 11ms/step -
accuracy: 0.5485 - loss: 0.6947 - val_accuracy: 0.5714 - val_loss: 0.6930
Epoch 8/50
13/13           0s 9ms/step -
accuracy: 0.5764 - loss: 0.6901 - val_accuracy: 0.5714 - val_loss: 0.6930
Epoch 9/50
13/13           0s 9ms/step -
accuracy: 0.5045 - loss: 0.6957 - val_accuracy: 0.4286 - val_loss: 0.6932
Epoch 10/50
13/13           0s 9ms/step -
accuracy: 0.4567 - loss: 0.6955 - val_accuracy: 0.4286 - val_loss: 0.6934
Epoch 11/50
13/13           0s 12ms/step -
accuracy: 0.4894 - loss: 0.7007 - val_accuracy: 0.4286 - val_loss: 0.6934
Epoch 12/50
13/13           0s 8ms/step -
accuracy: 0.4735 - loss: 0.6981 - val_accuracy: 0.4286 - val_loss: 0.6935
Epoch 13/50
13/13           0s 8ms/step -
accuracy: 0.4586 - loss: 0.6957 - val_accuracy: 0.4286 - val_loss: 0.6934
Epoch 14/50
13/13           0s 9ms/step -
accuracy: 0.4422 - loss: 0.6932 - val_accuracy: 0.4286 - val_loss: 0.6935
```

```
Epoch 15/50
13/13            0s 12ms/step -
accuracy: 0.4574 - loss: 0.6956 - val_accuracy: 0.4286 - val_loss: 0.6935
Epoch 16/50
13/13            0s 9ms/step -
accuracy: 0.4702 - loss: 0.6976 - val_accuracy: 0.4286 - val_loss: 0.6936
Epoch 17/50
13/13            0s 9ms/step -
accuracy: 0.4622 - loss: 0.6963 - val_accuracy: 0.4286 - val_loss: 0.6936
Epoch 18/50
13/13            0s 8ms/step -
accuracy: 0.4148 - loss: 0.6889 - val_accuracy: 0.4286 - val_loss: 0.6936
Epoch 19/50
13/13            0s 10ms/step -
accuracy: 0.4305 - loss: 0.6913 - val_accuracy: 0.4286 - val_loss: 0.6935
Epoch 20/50
13/13            0s 8ms/step -
accuracy: 0.4181 - loss: 0.6894 - val_accuracy: 0.4286 - val_loss: 0.6934
Epoch 21/50
13/13            0s 8ms/step -
accuracy: 0.4374 - loss: 0.6924 - val_accuracy: 0.4286 - val_loss: 0.6932
Epoch 22/50
13/13            0s 8ms/step -
accuracy: 0.4231 - loss: 0.6878 - val_accuracy: 0.5714 - val_loss: 0.6931
Epoch 23/50
13/13            0s 10ms/step -
accuracy: 0.5418 - loss: 0.6958 - val_accuracy: 0.5714 - val_loss: 0.6930
Epoch 24/50
13/13            0s 8ms/step -
accuracy: 0.5398 - loss: 0.6961 - val_accuracy: 0.5714 - val_loss: 0.6928
Epoch 25/50
13/13            0s 8ms/step -
accuracy: 0.6044 - loss: 0.6854 - val_accuracy: 0.5714 - val_loss: 0.6927
Epoch 26/50
13/13            0s 8ms/step -
accuracy: 0.5309 - loss: 0.6977 - val_accuracy: 0.5714 - val_loss: 0.6928
Epoch 27/50
13/13            0s 11ms/step -
accuracy: 0.5673 - loss: 0.6916 - val_accuracy: 0.5714 - val_loss: 0.6927
Epoch 28/50
13/13            0s 8ms/step -
accuracy: 0.5350 - loss: 0.6970 - val_accuracy: 0.5714 - val_loss: 0.6926
Epoch 29/50
13/13            0s 9ms/step -
accuracy: 0.5703 - loss: 0.6910 - val_accuracy: 0.5714 - val_loss: 0.6925
Epoch 30/50
13/13            0s 8ms/step -
accuracy: 0.5265 - loss: 0.6986 - val_accuracy: 0.5714 - val_loss: 0.6926
```

```
Epoch 31/50
13/13            0s 10ms/step -
accuracy: 0.5751 - loss: 0.6902 - val_accuracy: 0.5714 - val_loss: 0.6925
Epoch 32/50
13/13            0s 9ms/step -
accuracy: 0.5462 - loss: 0.6952 - val_accuracy: 0.5714 - val_loss: 0.6924
Epoch 33/50
13/13            0s 8ms/step -
accuracy: 0.5454 - loss: 0.6953 - val_accuracy: 0.5714 - val_loss: 0.6923
Epoch 34/50
13/13            0s 9ms/step -
accuracy: 0.5356 - loss: 0.6971 - val_accuracy: 0.5714 - val_loss: 0.6922
Epoch 35/50
13/13            0s 9ms/step -
accuracy: 0.5921 - loss: 0.6871 - val_accuracy: 0.5714 - val_loss: 0.6923
Epoch 36/50
13/13            0s 9ms/step -
accuracy: 0.5080 - loss: 0.7019 - val_accuracy: 0.5714 - val_loss: 0.6926
Epoch 37/50
13/13            0s 9ms/step -
accuracy: 0.5537 - loss: 0.6939 - val_accuracy: 0.5714 - val_loss: 0.6928
Epoch 38/50
13/13            0s 10ms/step -
accuracy: 0.5515 - loss: 0.6942 - val_accuracy: 0.5714 - val_loss: 0.6928
Epoch 39/50
13/13            0s 8ms/step -
accuracy: 0.5641 - loss: 0.6921 - val_accuracy: 0.5714 - val_loss: 0.6929
Epoch 40/50
13/13            0s 9ms/step -
accuracy: 0.5372 - loss: 0.6966 - val_accuracy: 0.5714 - val_loss: 0.6931
Epoch 41/50
13/13            0s 10ms/step -
accuracy: 0.6029 - loss: 0.6871 - val_accuracy: 0.4286 - val_loss: 0.6933
Epoch 42/50
13/13            0s 9ms/step -
accuracy: 0.4544 - loss: 0.6951 - val_accuracy: 0.4286 - val_loss: 0.6936
Epoch 43/50
13/13            0s 8ms/step -
accuracy: 0.4811 - loss: 0.6992 - val_accuracy: 0.4286 - val_loss: 0.6938
Epoch 44/50
13/13            0s 12ms/step -
accuracy: 0.4098 - loss: 0.6882 - val_accuracy: 0.4286 - val_loss: 0.6937
Epoch 45/50
13/13            0s 9ms/step -
accuracy: 0.4385 - loss: 0.6926 - val_accuracy: 0.4286 - val_loss: 0.6939
Epoch 46/50
13/13            0s 8ms/step -
accuracy: 0.4247 - loss: 0.6905 - val_accuracy: 0.4286 - val_loss: 0.6940
```

```
Epoch 47/50
13/13            0s 8ms/step -
accuracy: 0.4447 - loss: 0.6935 - val_accuracy: 0.4286 - val_loss: 0.6939
Epoch 48/50
13/13            0s 8ms/step -
accuracy: 0.4379 - loss: 0.6925 - val_accuracy: 0.4286 - val_loss: 0.6938
Epoch 49/50
13/13            0s 9ms/step -
accuracy: 0.4662 - loss: 0.6968 - val_accuracy: 0.4286 - val_loss: 0.6937
Epoch 50/50
13/13            0s 8ms/step -
accuracy: 0.4611 - loss: 0.6961 - val_accuracy: 0.4286 - val_loss: 0.6935
```

[56]:
```python
# 8. Evaluate
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')
```

```
2/2           0s 25ms/step -
accuracy: 0.5476 - loss: 0.6929
Test Accuracy: 0.52
```

0.4 Model Evaluation (Confusion Matrix and Classification Report)

After training, it's important to evaluate how the model performs beyond just accuracy, especially since we have class imbalance. We'll use a confusion matrix and a classification report (precision, recall, f1-score).

[57]:
```python
# Predict on test set
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)  # Threshold of 0.5
```

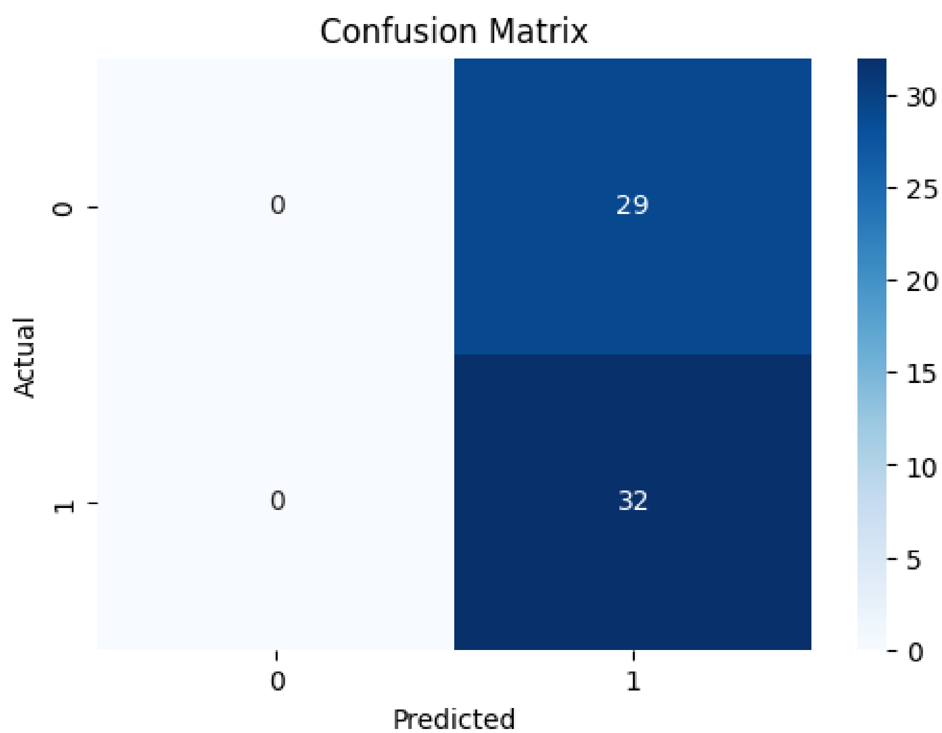
```
WARNING:tensorflow:5 out of the last 5 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x000001B4E2454B80> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/2           0s
57ms/stepWARNING:tensorflow:6 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x000001B4E2454B80> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
```

```
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/2              0s 67ms/step
```

[58]:

```python
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



[59]:

```python
# Classification Report
print("Classification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support
```

```
           0       0.00      0.00      0.00        29
           1       0.52      1.00      0.69        32

    accuracy                           0.52        61
   macro avg       0.26      0.50      0.34        61
weighted avg       0.28      0.52      0.36        61

C:\Users\anves\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n
2kfra8p0\LocalCache\local-packages\Python312\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\anves\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n
2kfra8p0\LocalCache\local-packages\Python312\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\anves\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n
2kfra8p0\LocalCache\local-packages\Python312\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

0.5 Conclusion:

- Precision for Class 0 is currently 0, indicating that the model is failing to predict the majority class ("No Disease").
- Recall for Class 1 is 1.0, which is excellent, but it comes at the cost of a high number of false positives (low precision for class 1).
- This is done to ensure that all patients with heart disease get identified even if there are false positives, as **saving a life matters more than guessing it wrong.**