CCE2-Disease Prediction Using Machine Learning

Submitted in partial fulfillment of the requirements of the
degree of **Bachelor's in Engineering**

for

AIML in Healthcare Domain

Submitted by

POOJA YADAV/ TY-09-67
MANSI VAISHYA/ TY-09-64
NEHA PALVI/ TY-09-43

COMPUTER ENGINEERING DEPARTMENT

SHAH & ANCHOR KUTCHHI ENGINEERING COLLEGE
(An Autonomous Institute Affiliates to University of Mumbai)

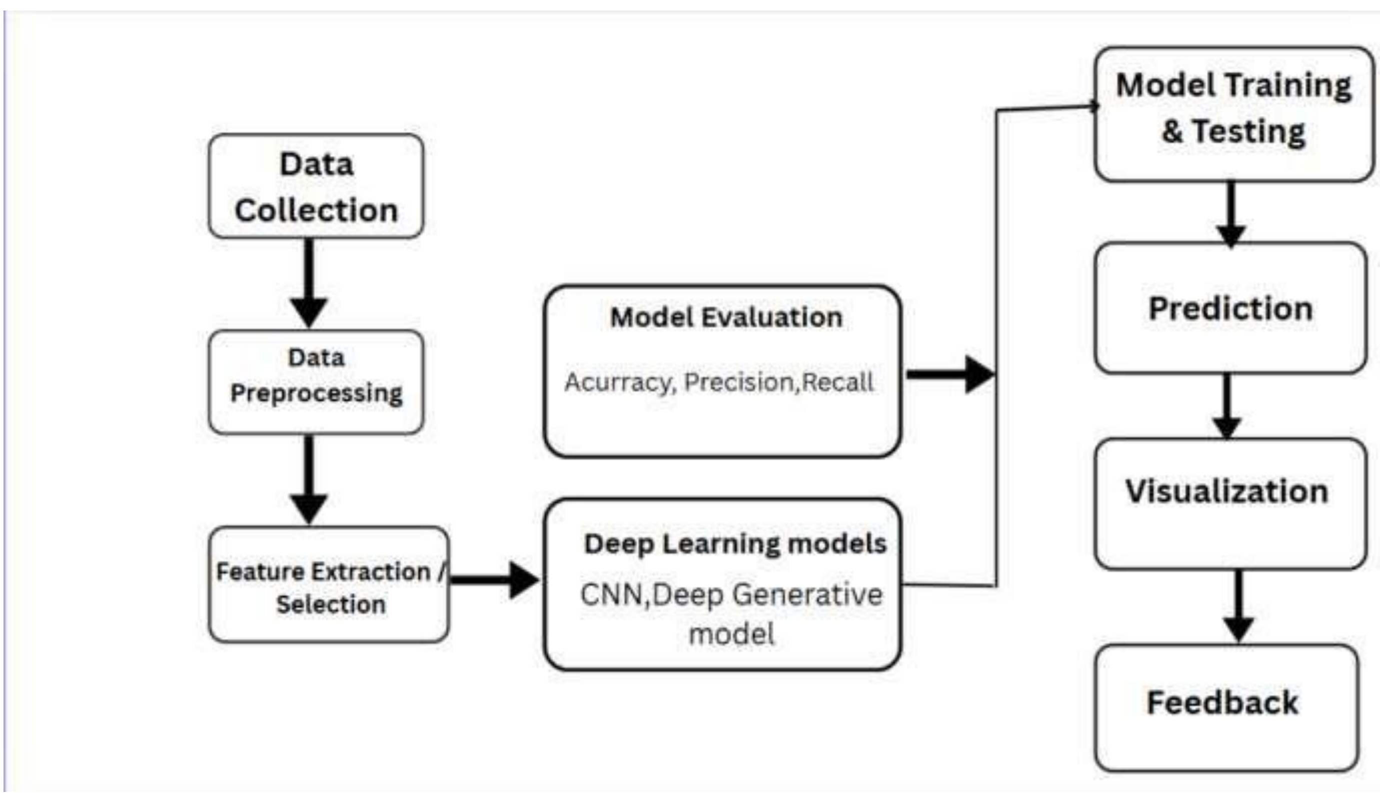
MUMBAI-400 088, MAHARASHTRA (INDIA)

Academic your 2024-25

Abstract:

Artificial Intelligence (AI) and Machine Learning (ML) are revolutionizing the healthcare industry by enhancing diagnostic accuracy, personalizing treatment, and improving patient outcomes. AI/ML systems enable machines to analyze complex medical data, identify patterns, and assist healthcare professionals in decision-making. This integration of data science, medicine, and computational intelligence has led to major advancements in disease detection, drug discovery, and patient monitoring. From image-based diagnostics using deep learning to predictive analytics in public health, AI/ML is transforming every aspect of modern medicine. However, challenges such as data privacy, model bias, and regulatory hurdles remain critical considerations for widespread adoption.

Architecture Diagra



Methodology Explanation:

The Disease Prediction System using AI/ML shown in the diagram demonstrates how different roles — data scientists, developers, healthcare experts, testers, and end-users — interact with various components and databases throughout the model development and deployment life cycle.

1. Data Scientist / ML Researcher

- **Responsibilities:** design the prediction problem (labels, outcomes), choose algorithms, create features, train and validate models.
- **Use the Data Catalog & Feature Store:** access curated datasets, engineered features, and metadata.
- **Store experiments & documentation:** model designs, notebooks, hyperparameters and evaluation results go into the **Model Registry / Experiment Tracking** system.

---

2. ML Engineer / Developer

- **Responsibilities:** productionize models, build training pipelines, create inference services (APIs), and integrate ML models into apps or hospital systems.
- **Use the Code Repository & CI/CD:** code for preprocessing, model training, deployment scripts and infrastructure-as-code are kept in the **Code Repository** (version control).
- **Deploy models to the Serving Environment:** packages are deployed to the **Inference API / Model Serving** with monitoring hooks.

---

3. Clinician / Domain Expert

- **Responsibilities:** provide clinical labels, validate feature relevance, review model outputs, and ensure clinical usability and safety.
- **Use the Knowledge Base & Annotation Tools:** provide annotations, create and approve clinical definitions, and keep the **Clinical Guidelines Repository** updated.
- **Sign off on high-risk decisions:** authorize models for use in triage, alerts, or diagnostic assistance.

---

4. QA / Tester

- **Responsibilities:** test data pipelines, model behavior, integration points, and UI for edge cases and robustness.
- **Test on staging builds:** use anonymized or synthetic data in the **Staging Environment** to validate performance and safety.
- **Log issues:** bugs, performance regressions and incorrect model outputs are tracked in the **Issue/Bug Database**.

---

5. Patient / End-User

- **Interaction:** patients (or clinicians acting on behalf of patients) interact with the interface (app, EHR integration, dashboard) and receive predictions, risk scores, or recommendations.
- **Feedback:** patient outcomes and clinician feedback flow back to the system to improve models (with consent and privacy controls).

- **Data Collection:**
  The process begins with collecting patient-related data such as symptoms, medical history, lab test results, and demographic information from various sources like hospitals or public datasets.
- **Data Preprocessing:**
  The collected data often contains noise, missing values, or inconsistencies. This stage cleans and standardizes the data through techniques such as normalization, data balancing, and handling of missing values.
- **Feature Extraction / Selection:**
  Important features (e.g., age, blood pressure, glucose level) are extracted or selected from the dataset to train the model efficiently. This helps reduce dimensionality and improve model performance.
- **Deep Learning Models:**
  Deep learning architectures like CNN (Convolutional Neural Network) or Deep Generative Models are used to analyze the processed data and learn complex patterns that can predict disease outcomes accurately.
- **Model Training & Testing:**
  The selected models are trained using the prepared dataset and later tested on unseen data to evaluate their ability to predict diseases correctly.
- **Model Evaluation:**
  The performance of the model is measured using metrics such as Accuracy, Precision, and Recall to ensure its reliability and robustness.
- **Prediction:**
  Once evaluated, the trained model predicts whether a patient is likely to have a particular disease based on their input data.
- **Visualization:**
  The prediction results are visualized using graphs, dashboards, or reports, making it easier for doctors and users to interpret the outcomes.
- **Feedback:**
  Feedback from healthcare experts or users is collected to refine the system, update the dataset, and retrain the model for improved accuracy in future predictions.

Result Analysis / Coding:

```python
# Cell 1 - Imports & dataset
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve, classification_report, confusion_matrix
import joblib
import matplotlib.pyplot as plt

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
cols = ["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction","Age","Outcome"]
df = pd.read_csv(url, header=None, names=cols)

df_clean = df.copy()
zero_cols = ["Glucose","BloodPressure","SkinThickness","Insulin","BMI"]
for c in zero_cols:
    med = df_clean.loc[df_clean[c] != 0, c].median()
    df_clean.loc[df_clean[c] == 0, c] = med

X = df_clean.drop("Outcome", axis=1).values
y = df_clean["Outcome"].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)
```

```python
# Cell 2 - RandomForest training & evaluation
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train_s, y_train)
y_pred_rf = rf.predict(X_test_s)
y_proba_rf = rf.predict_proba(X_test_s)[:,1]
acc_rf = accuracy_score(y_test, y_pred_rf)
auc_rf = roc_auc_score(y_test, y_proba_rf)

print("RandomForest accuracy:", acc_rf)
print("RandomForest ROC AUC:", auc_rf)
print(classification_report(y_test, y_pred_rf))
```

```
RandomForest accuracy: 0.7402597402597403
RandomForest ROC AUC: 0.8161111111111112
              precision    recall  f1-score   support

           0       0.78      0.84      0.81       100
           1       0.65      0.56      0.60        54

    accuracy                           0.74       154
   macro avg       0.71      0.70      0.70       154
weighted avg       0.73      0.74      0.73       154
```

```python
# Cell 3 - Prepare data for 1D-CNN (reshape)
X_train_cnn = X_train_s.reshape((-1, X_train_s.shape[1], 1))
X_test_cnn = X_test_s.reshape((-1, X_test_s.shape[1], 1))
```

```python
# Cell 4 - 1D CNN model (Keras) training
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, GlobalAveragePooling1D, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

tf.random.set_seed(42)

cnn = Sequential([
    Conv1D(32, kernel_size=2, activation='relu', input_shape=(X_train_cnn.shape[1], 1)),
    Conv1D(16, kernel_size=2, activation='relu'),
    GlobalAveragePooling1D(),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
es = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
history = cnn.fit(X_train_cnn, y_train, validation_split=0.15, epochs=100, batch_size=32, callbacks=[es], verbose=2)

y_proba_cnn = cnn.predict(X_test_cnn).ravel()
y_pred_cnn = (y_proba_cnn >= 0.5).astype(int)
```

```python
print("CNN accuracy:", acc_cnn)
print("CNN ROC AUC:", auc_cnn)
print(classification_report(y_test, y_pred_cnn))
```

```
17/17 - 0s - 8ms/step - accuracy: 0.7735 - loss: 0.4857 - val_accuracy: 0.7634 - val_loss: 0.4556
Epoch 79/100
17/17 - 0s - 11ms/step - accuracy: 0.7658 - loss: 0.4785 - val_accuracy: 0.7634 - val_loss: 0.4580
Epoch 80/100
17/17 - 0s - 15ms/step - accuracy: 0.7658 - loss: 0.4778 - val_accuracy: 0.7742 - val_loss: 0.4566
Epoch 81/100
17/17 - 0s - 8ms/step - accuracy: 0.7850 - loss: 0.4671 - val_accuracy: 0.7634 - val_loss: 0.4545
Epoch 82/100
17/17 - 0s - 8ms/step - accuracy: 0.7716 - loss: 0.4743 - val_accuracy: 0.7634 - val_loss: 0.4549
Epoch 83/100
17/17 - 0s - 8ms/step - accuracy: 0.7774 - loss: 0.4707 - val_accuracy: 0.7634 - val_loss: 0.4550
Epoch 84/100
17/17 - 0s - 8ms/step - accuracy: 0.7774 - loss: 0.4787 - val_accuracy: 0.7742 - val_loss: 0.4545
Epoch 85/100
17/17 - 0s - 9ms/step - accuracy: 0.7678 - loss: 0.4723 - val_accuracy: 0.7742 - val_loss: 0.4544
Epoch 86/100
17/17 - 0s - 8ms/step - accuracy: 0.7678 - loss: 0.4767 - val_accuracy: 0.7742 - val_loss: 0.4538
Epoch 87/100
17/17 - 0s - 8ms/step - accuracy: 0.7774 - loss: 0.4692 - val_accuracy: 0.7634 - val_loss: 0.4533
Epoch 88/100
17/17 - 0s - 8ms/step - accuracy: 0.7812 - loss: 0.4770 - val_accuracy: 0.7742 - val_loss: 0.4533
Epoch 89/100
17/17 - 0s - 8ms/step - accuracy: 0.7697 - loss: 0.4709 - val_accuracy: 0.7634 - val_loss: 0.4566
Epoch 90/100
17/17 - 0s - 8ms/step - accuracy: 0.7754 - loss: 0.4727 - val_accuracy: 0.7742 - val_loss: 0.4549
Epoch 91/100
17/17 - 0s - 8ms/step - accuracy: 0.7889 - loss: 0.4580 - val_accuracy: 0.7634 - val_loss: 0.4541
```

```
17/17 - 0s - 9ms/step - accuracy: 0.7889 - loss: 0.4611 - val_accuracy: 0.7634 - val_loss: 0.4544
Epoch 96/100
17/17 - 0s - 8ms/step - accuracy: 0.7716 - loss: 0.4648 - val_accuracy: 0.7742 - val_loss: 0.4511
Epoch 97/100
17/17 - 0s - 8ms/step - accuracy: 0.7754 - loss: 0.4698 - val_accuracy: 0.7634 - val_loss: 0.4528
Epoch 98/100
17/17 - 0s - 15ms/step - accuracy: 0.7812 - loss: 0.4645 - val_accuracy: 0.7742 - val_loss: 0.4511
Epoch 99/100
17/17 - 0s - 16ms/step - accuracy: 0.7774 - loss: 0.4655 - val_accuracy: 0.7742 - val_loss: 0.4537
Epoch 100/100
17/17 - 0s - 16ms/step - accuracy: 0.7850 - loss: 0.4696 - val_accuracy: 0.7742 - val_loss: 0.4507
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
CNN accuracy: 0.6948051948051948
CNN ROC AUC: 0.7862962962962964
              precision    recall  f1-score   support

           0       0.79      0.72      0.75       100
           1       0.56      0.65      0.60        54

    accuracy                           0.69       154
   macro avg       0.67      0.68      0.68       154
weighted avg       0.71      0.69      0.70       154
```

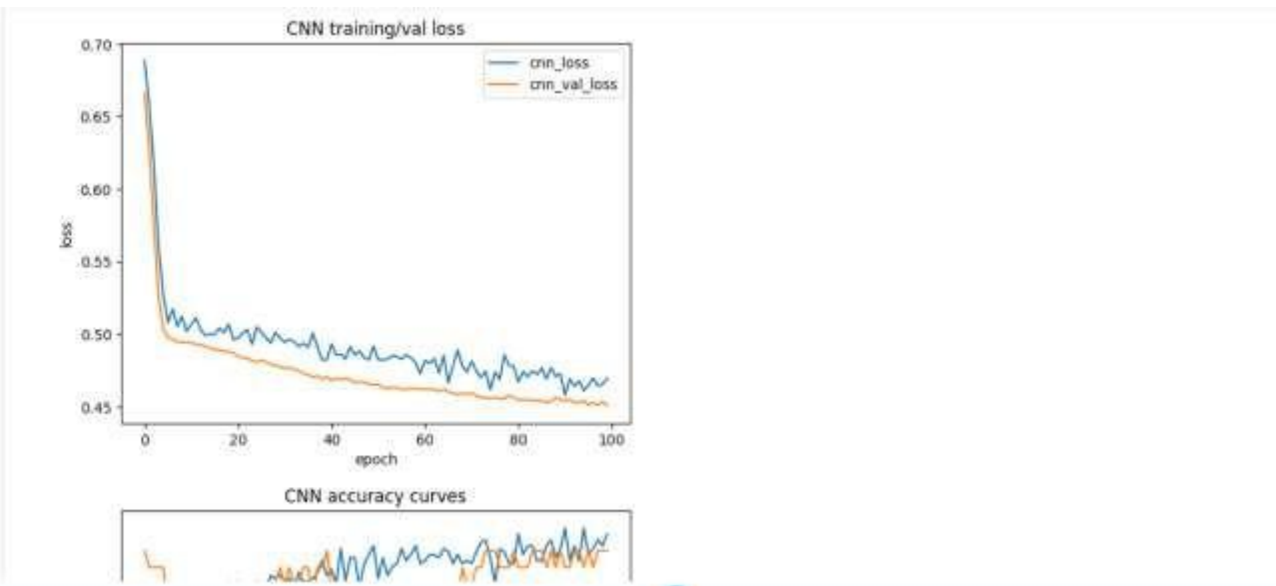
[5]
✓ 16s

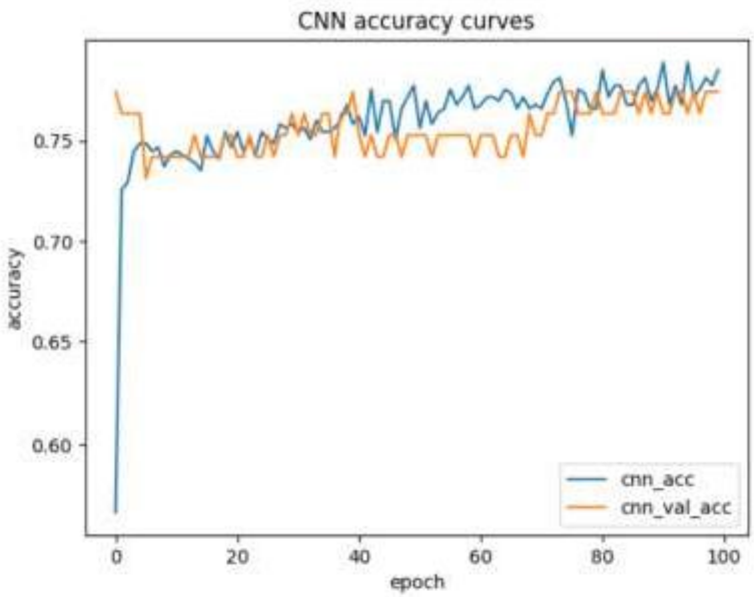

```python
# Cell 5 - Autoencoder to get latent representations, then RandomForest on latent space
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

input_dim = X_train_s.shape[1]
encoding_dim = 4
```

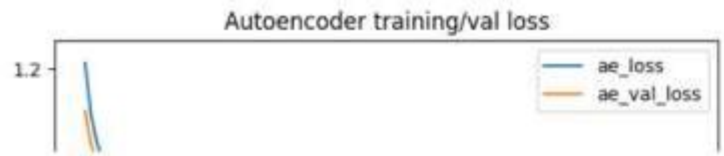

```python
# Cell 5 - Autoencoder to get latent representations, then RandomForest on latent space
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

input_dim = X_train_s.shape[1]
encoding_dim = 4

input_layer = Input(shape=(input_dim,))
encoded = Dense(8, activation='relu')(input_layer)
encoded = Dense(encoding_dim, activation='relu')(encoded)
decoded = Dense(8, activation='relu')(encoded)
decoded = Dense(input_dim, activation='linear')(decoded)

autoencoder = Model(inputs=input_layer, outputs=decoded)
encoder = Model(inputs=input_layer, outputs=encoded)

autoencoder.compile(optimizer='adam', loss='mse')
es2 = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
hist_ae = autoencoder.fit(X_train_s, X_train_s, validation_split=0.15, epochs=100, batch_size=32, callbac

X_train_latent = encoder.predict(X_train_s)
X_test_latent = encoder.predict(X_test_s)

rf_latent = RandomForestClassifier(n_estimators=200, random_state=42)
rf_latent.fit(X_train_latent, y_train)
y_pred_ae = rf_latent.predict(X_test_latent)
y_proba_ae = rf_latent.predict_proba(X_test_latent)[:,1]
acc_ae = accuracy_score(y_test, y_pred_ae)
auc_ae = roc_auc_score(y_test, y_proba_ae)
```

```
# Cell 6 - Plot training curves for CNN and Autoencoder
plt.figure()
plt.plot(history.history['loss'], label='cnn_loss')
plt.plot(history.history['val_loss'], label='cnn_val_loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.title('CNN training/val loss')
plt.show()

plt.figure()
plt.plot(history.history.get('accuracy', []), label='cnn_acc')
plt.plot(history.history.get('val_accuracy', []), label='cnn_val_acc')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend()
plt.title('CNN accuracy curves')
plt.show()

plt.figure()
plt.plot(hist_ae.history['loss'], label='ae_loss')
plt.plot(hist_ae.history['val_loss'], label='ae_val_loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.title('Autoencoder training/val loss')
plt.show()
```

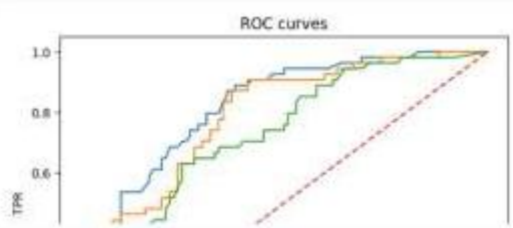
Results – Graph or Output Snapshots:



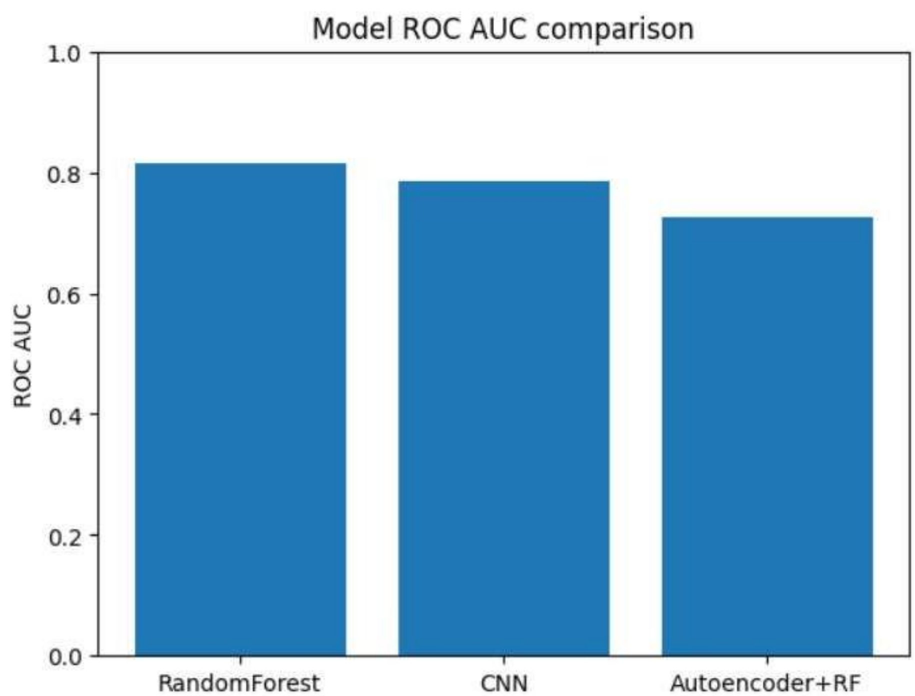





```
# Cell 7 - ROC curves for all three
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_proba_rf)
fpr_cnn, tpr_cnn, _ = roc_curve(y_test, y_proba_cnn)
fpr_ae, tpr_ae, _ = roc_curve(y_test, y_proba_ae)

plt.figure()
plt.plot(fpr_rf, tpr_rf, label=f'RF AUC={auc_rf:.3f}')
plt.plot(fpr_cnn, tpr_cnn, label=f'CNN AUC={auc_cnn:.3f}')
plt.plot(fpr_ae, tpr_ae, label=f'AE+RF AUC={auc_ae:.3f}')
plt.plot([0,1],[0,1], linestyle='--')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.title('ROC curves')
plt.show()
```



```python
# Cell 8 - Bar chart comparing AUCs and select best
aucs = {'RandomForest': auc_rf, 'CNN': auc_cnn, 'Autoencoder+RF': auc_ae}
names = list(aucs.keys())
vals = [aucs[n] for n in names]

plt.figure()
plt.bar(range(len(vals)), vals)
plt.xticks(range(len(vals)), names)
plt.ylabel('ROC AUC')
plt.title('Model ROC AUC comparison')
plt.ylim(0,1)
plt.show()

best_name = max(aucs, key=aucs.get)
best_auc = aucs[best_name]
print("Best model:", best_name, "with ROC AUC =", best_auc)
```




```
Best model: RandomForest with ROC AUC = 0.8161111111111112
```

```python
# Cell 9 - Save best model and scaler
if best_name == 'RandomForest':
    joblib.dump(rf, "best_model.pkl")
elif best_name == 'CNN':
    cnn.save("best_cnn_model.h5")
elif best_name == 'Autoencoder+RF':
    joblib.dump(rf_latent, "best_model.pkl")
    joblib.dump(encoder, "encoder_model.pkl")
joblib.dump(scaler, "scaler.pkl")
print("Saved best model and scaler.")
```

```
Saved best model and scaler.
```

```python
# Cell 10 - Example predict function using the selected best model
import numpy as np
def predict_sample(sample_array):  # sample_array is 1D list/array of 8 raw feature values
    x = np.array(sample_array).reshape(1,-1)
    x_s = scaler.transform(x)
    if best_name == 'RandomForest':
        proba = rf.predict_proba(x_s)[:,1][0]
        pred = int(proba >= 0.5)
    elif best_name == 'CNN':
        x_c = x_s.reshape((1, x_s.shape[1], 1))
        proba = cnn.predict(x_c).ravel()[0]
        pred = int(proba >= 0.5)
    else:
        lat = encoder.predict(x_s)
        proba = rf_latent.predict_proba(lat)[:,1][0]
```