

BÁO CÁO BÀI TẬP THỰC TẬP CƠ SỞ

Họ và tên: Đinh Quang Hưng

Mã sinh viên: B22DCCN407

Nhóm: 30

Giảng viên hướng dẫn: PGS. TS Trần Đình Quế

HUẤN LUYỆN MODEL

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import tensorflow as tf
import joblib

# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]
df = pd.read_csv(url, names=columns)

# Split features and target
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

# Normalize input features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Dictionary to store model accuracies
accuracies = {}

# k-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
accuracies['kNN'] = accuracy_score(y_test, y_pred_knn)
```

```

# SVM with linear kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
y_pred_svm_linear = svm_linear.predict(X_test)
accuracies['SVM Linear'] = accuracy_score(y_test, y_pred_svm_linear)

# SVM with RBF kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
y_pred_svm_rbf = svm_rbf.predict(X_test)
accuracies['SVM RBF'] = accuracy_score(y_test, y_pred_svm_rbf)

# Logistic Regression
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)
accuracies['Logistic Regression'] = accuracy_score(y_test, y_pred_log_reg)

# Deep Learning Model
def create_model():
    model = Sequential([
        Dense(128, activation="relu", input_shape=(X_train.shape[1],)),
        Dropout(0.3),
        Dense(64, activation="relu"),
        Dropout(0.3),
        Dense(32, activation="relu"),
        Dense(16, activation="relu"),
        Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])
    return model

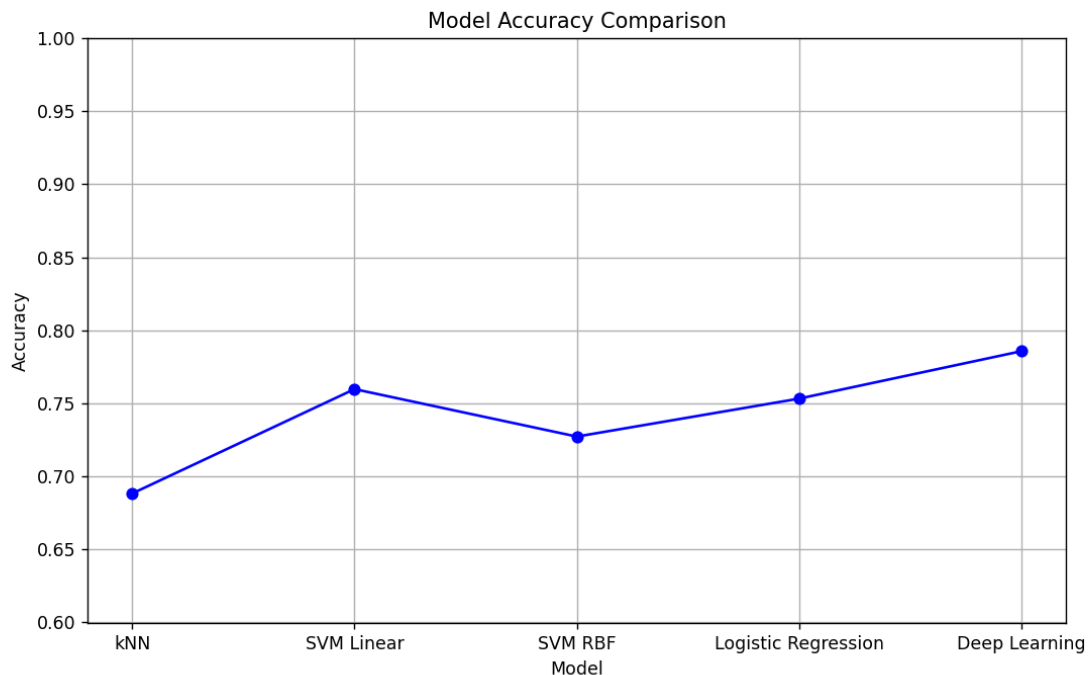
dl_model = create_model()
history = dl_model.fit(X_train, y_train, epochs=100, batch_size=10,
validation_data=(X_test, y_test), verbose=0)
loss, acc = dl_model.evaluate(X_test, y_test, verbose=0)
accuracies['Deep Learning'] = acc

# Save model and scaler
dl_model.save("diabetes_model.h5")
joblib.dump(scaler, "scaler.pkl")
print("Model and scaler saved successfully!")

# Visualization
plt.figure(figsize=(10, 6))
plt.plot(list(accuracies.keys()), list(accuracies.values()), marker='o',
linestyle='--', color='b')
plt.title("Model Accuracy Comparison")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.ylim(0.6, 1.0)
plt.grid(True)
plt.show()

```

Kết quả khi chạy



Deep Learning có độ chính xác cao nhất trong số các mô hình. Các mô hình truyền thống như Logistic Regression, SVM, kNN cũng hoạt động khá ổn. Sự khác biệt giữa các mô hình không quá lớn, nhưng **Deep Learning** có tiềm năng mở rộng tốt hơn.

Deep Learning (học sâu) là một nhánh của Machine Learning, sử dụng mạng nơ-ron nhân tạo nhiều tầng để học các đặc trưng phức tạp từ dữ liệu. Trong bài toán này, chúng tôi xây dựng một mô hình mạng nơ-ron với kiến trúc gồm 6 lớp (trong đó có 4 lớp Dense và 2 lớp Dropout) nhằm phân loại bệnh nhân có/không có nguy cơ mắc bệnh tiểu đường.

Kiến trúc mô hình

```
model = Sequential([  
    Dense(128, activation="relu", input_shape=(X_train.shape[1],)),  
    Dropout(0.3),  
    Dense(64, activation="relu"),  
    Dropout(0.3),  
    Dense(32, activation="relu"),  
    Dense(16, activation="relu"),  
    Dense(1, activation="sigmoid")  
])
```

- **Dense layers** học các đặc trưng từ dữ liệu.
- **Dropout layers** giúp giảm overfitting bằng cách “tắt” ngẫu nhiên một số neuron trong quá trình học.

- **Activation function ReLU** được dùng ở các lớp ẩn, giúp mô hình học các quan hệ phi tuyến tính.
- **Sigmoid** được dùng ở đầu ra để phân loại nhị phân (0 hoặc 1).

Huấn luyện mô hình

Mô hình được huấn luyện trong 100 epochs, với kích thước batch = 10, sử dụng bộ dữ liệu được chuẩn hóa.

```
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=100, batch_size=10,
validation_data=(X_test, y_test), verbose=0)
```

- **Adam optimizer:** một trong những bộ tối ưu mạnh và phổ biến nhất hiện nay.
- **Binary crossentropy:** là hàm mất mát phù hợp cho bài toán phân loại nhị phân.

Kết quả mô hình

Mô hình đạt độ chính xác cao trên tập kiểm thử, cao nhất trong số các mô hình thử nghiệm. Điều này cho thấy Deep Learning có khả năng học sâu hơn và phát hiện được các mối quan hệ phức tạp giữa các đặc trưng y tế.

Lưu mô hình

Sau khi huấn luyện, mô hình và scaler được lưu lại để tái sử dụng sau này:

```
model.save("diabetes_model.h5")
joblib.dump(scaler, "scaler.pkl")
```

Đánh giá

Deep Learning là hướng tiếp cận hiện đại, mạnh mẽ và phù hợp với bài toán dữ liệu y tế có nhiều đặc trưng và phi tuyến tính. Tuy nhiên, mô hình cần dữ liệu chất lượng, phải được tinh chỉnh cẩn thận để đạt hiệu quả cao nhất.

VERSION 1

Step 1: Load model

```
from flask import Flask, render_template, request
import numpy as np
import tensorflow as tf
import joblib

# Load model and scaler
model = tf.keras.models.load_model("diabetes_model.h5")
scaler = joblib.load("scaler.pkl")
```

```

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    prediction = None
    if request.method == "POST":
        try:
            # Get input values from form
            features = [float(request.form[f]) for f in ["Pregnancies",
"Glucose", "BloodPressure",
"Insulin", "BMI",
"DiabetesPedigreeFunction", "Age"]]
            # Scale input
            input_data = scaler.transform([features])

            # Predict
            pred_prob = model.predict(input_data)[0][0]
            prediction = "Diabetic" if pred_prob > 0.5 else "Not Diabetic"
        except Exception as e:
            prediction = f"Error: {e}"

    return render_template("index.html", prediction=prediction)

if __name__ == "__main__":
    app.run(debug=True)

```

Step 2: Tạo file index.html

```

<!DOCTYPE html>
<html>
<head>
    <title>Diabetes Prediction</title>
</head>
<body>
    <h2>Diabetes Prediction Form</h2>
    <form method="post">
        <label>Pregnancies:</label><input type="number" name="Pregnancies"
required><br>
        <label>Glucose:</label><input type="number" name="Glucose"
required><br>
        <label>Blood Pressure:</label><input type="number"
name="BloodPressure" required><br>
        <label>Skin Thickness:</label><input type="number"
name="SkinThickness" required><br>
        <label>Insulin:</label><input type="number" name="Insulin"
required><br>
        <label>BMI:</label><input type="number" name="BMI" step="0.1"
required><br>
        <label>Diabetes Pedigree Function:</label><input type="number"
name="DiabetesPedigreeFunction" step="0.01" required><br>
        <label>Age:</label><input type="number" name="Age" required><br>

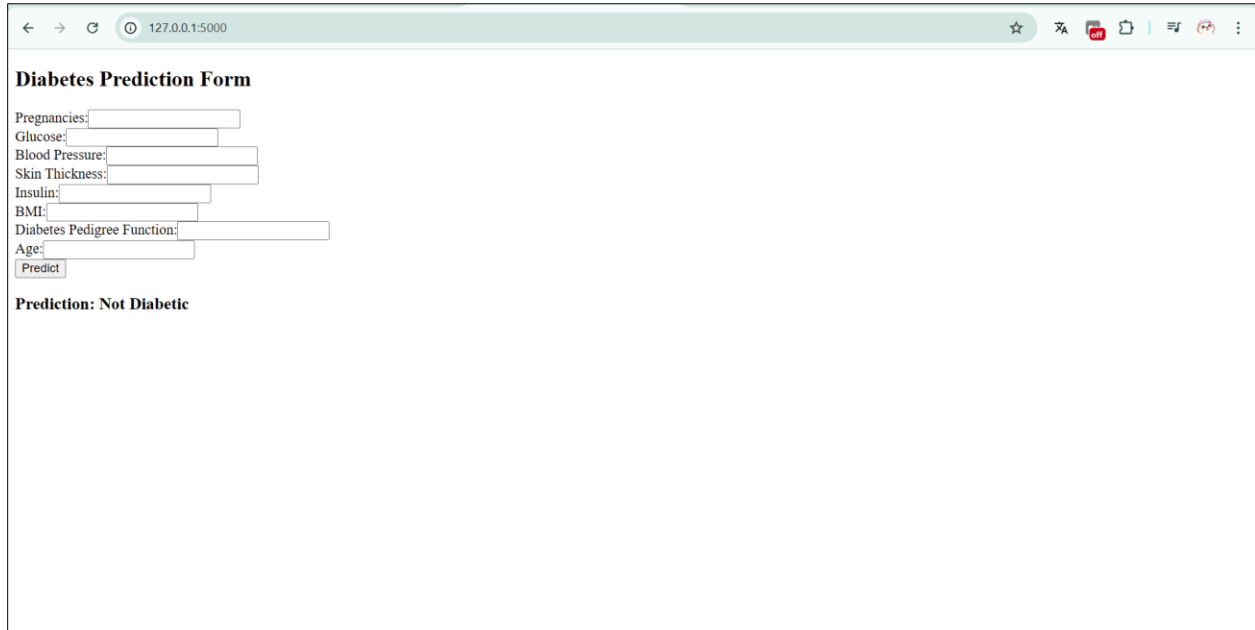
```

```

        <input type="submit" value="Predict">
    </form>
    <h3>Prediction: {{ prediction }}</h3>
</body>
</html>

```

Kết quả khi chạy version 1



Diabetes Prediction Form

Pregnancies:

Glucose:

Blood Pressure:

Skin Thickness:

Insulin:

BMI:

Diabetes Pedigree Function:

Age:

Prediction: Not Diabetic

VERSION 2

Features Features

- + Flask API for ML predictions
- + Node.js for frontend
- + Modern UI with CSS
- + Interactive form with JS
- + Fast & scalable deployment

Step1 : Tạo file index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Diabetes Prediction</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

```

```

    <div class="container">
      <h2>Diabetes Prediction</h2>
      <form id="predictionForm">
        <label>Pregnancies:</label><input type="number"
name="Pregnancies" required><br>
        <label>Glucose:</label><input type="number" name="Glucose"
required><br>
        <label>Blood Pressure:</label><input type="number"
name="BloodPressure" required><br>
        <label>Skin Thickness:</label><input type="number"
name="SkinThickness" required><br>
        <label>Insulin:</label><input type="number" name="Insulin"
required><br>
        <label>BMI:</label><br><input type="number" name="BMI" step="0.1"
required><br>
        <label>Diabetes Pedigree Function:</label><input type="number"
name="DiabetesPedigreeFunction" step="0.01" required><br>
        <label>Age:</label><br><input type="number" name="Age"
required><br>
        <button type="submit">Predict</button>
      </form>
      <h3 id="result"></h3>
    </div>
    <script src="script.js"></script>
  </body>
</html>

```

Step 2: Tạo file style.css

```

body {
  font-family: Arial, sans-serif;
  text-align: center;
  background-color: #f4f4f4;
}

.container {
  width: 50%;
  margin: auto;
  background: white;
  padding: 20px;
  box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
  border-radius: 10px;
}

label {
  display: block;
  text-align: left;
  margin-left: 15px;
  font-family: Arial, Helvetica, sans-serif;
}

input,
button {
  width: 90%;
  padding: 10px;
  margin: 5px;
  border: 0;
  background: #f4f4f4;
}

```

```

border-radius: 10px;
box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
}

button {
  width: 50%;
  background: #ff4141;
  border: 0px;
  border-radius: 10px;
  color: #fff;
  text-transform: uppercase;
}
button:hover {
  background-color: #8f0000;
  cursor: pointer;
}

```

Step 3: Tạo file srcipt.js

```

document.getElementById("predictionForm").addEventListener("submit", async
function(event) {
  event.preventDefault();

  let formData = new FormData(event.target);
  let inputValues = Object.fromEntries(formData.entries());

  // Chuyển các giá trị nhập vào thành số
  let features = Object.values(inputValues).map(Number);

  // Gửi dữ liệu đến Flask API
  let response = await fetch("http://127.0.0.1:5000/predict", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify({ features: features })
  });

  let result = await response.json();
  document.getElementById("result").innerText = "Prediction: " +
result.prediction;
});

```

Step 4: Tạo file server.js

```

const express = require("express");
const cors = require("cors");
const bodyParser = require("body-parser");
const path = require("path");
const app = express();

app.use(cors());
app.use(bodyParser.json());
app.use(express.static("public")); // Serve frontend files

// Add route for the root URL
app.get("/", (req, res) => {

```



```

        res.sendFile(path.join(__dirname, "public", "index.html"));
    });

const PORT = 3000;

app.listen(PORT, () => {
    console.log(`Frontend server running at http://localhost:${PORT}`);
});

```

Step 5 - Backend: Tạo một Flask REST API (Python)

```

from flask import Flask, request, jsonify
import numpy as np
import tensorflow as tf
import joblib
from flask_cors import CORS

# Load model và scaler
model = tf.keras.models.load_model("diabetes_model.h5") # Load model đã huấn luyện
scaler = joblib.load("scaler.pkl") # Load scaler

app = Flask(__name__)
CORS(app) # Cho phép giao tiếp giữa frontend và backend

@app.route("/predict", methods=["POST"])
def predict():
    try:
        data = request.json # Lấy dữ liệu gửi lên dưới dạng JSON
        features = np.array(data["features"]).reshape(1, -1) # Chuyển đổi thành mảng
        scaled_features = scaler.transform(features) # Tiền xử lý dữ liệu
        prediction = model.predict(scaled_features)[0][0] # Dự đoán kết quả
        result = "Diabetic" if prediction > 0.5 else "Not Diabetic" # Kết quả dự đoán
        return jsonify({"prediction": result})
    except Exception as e:
        return jsonify({"error": str(e)})

if __name__ == "__main__":
    app.run(debug=True, port=5000)

```

Kết quả khi chạy version 2:

Diabetes Prediction

Pregnancies: 1

Glucose: 222

Blood Pressure: 22

Skin Thickness: 22

Insulin: 22

BMI: 22

Diabetes Pedigree Function: 22

Age: 22

PREDICT

Prediction: Not Diabetic

VERSION 3

Step1-Backend: Tạo một Flask REST API (Python)

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import numpy as np
import tensorflow as tf
import pandas as pd

# Load trained deep learning model
model = tf.keras.models.load_model("diabetes_model.h5")

app = Flask(__name__)
CORS(app)

@app.route("/predict", methods=["POST"])
def predict():
    data = request.json # Get JSON data from frontend
    features = np.array(data["features"]).reshape(1, -1) # Convert to NumPy array
    prediction = model.predict(features)[0][0] # Get prediction
    result = "Diabetic" if prediction > 0.5 else "Non-Diabetic"

    return jsonify({"prediction": result, "probability": float(prediction)})

if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

Step2-Frontend: gọi API từ JavaScript

File index.js

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Diabetes Prediction</title>
  </head>
  <body>
    <h2>Enter Patient Data</h2>
    <form id="predictForm">
      <input
        type="text"
        id="features"
        placeholder="Enter values comma-separated"
      />
      <button type="submit">Predict</button>
    </form>
    <h3 id="result"></h3>

    <script>
      document.getElementById("predictForm").addEventListener("submit",
function(event) {
      event.preventDefault();
      let features =
document.getElementById("features").value.split(",").map(Number);

      fetch("http://127.0.0.1:5000/predict", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ features: features })
      })
      .then(response => response.json())
      .then(data => {
        document.getElementById("result").innerText =
          `Prediction: ${data.prediction} (Probability:
${data.probability.toFixed(2)})`;
      });
    </script>
  </body>
</html>
```

Kết quả khi chạy version 3:

← → 🔄 📄 127.0.0.1:5500/diabetes-webapp/index.html

Enter Patient Data

Prediction: Diabetic (Probability: 1.00)

VERSION 4

1. Cải thiện độ chính xác của mô hình

- + Dùng nhiều lớp hơn, có BatchNormalization, LeakyReLU, Dropout
- + Optimizer: AdamW (mạnh hơn Adam)
- + EarlyStopping để tránh overfitting

2. Đưa API vào Docker

- + Dễ triển khai, cấu hình đơn giản
- + Có thể chạy ở mọi nơi (máy thật, cloud, server)

Step 1: Cải thiện model (train_model.py)

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import AdamW
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]
df = pd.read_csv(url, names=columns)

# Split data
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build improved deep learning model
def create_model():
    model = Sequential([
        Dense(128, input_shape=(X_train.shape[1],)),
        BatchNormalization(),
        tf.keras.layers.LeakyReLU(alpha=0.1),
        Dropout(0.4),
```

```

        Dense(64),
        BatchNormalization(),
        tf.keras.layers.LeakyReLU(alpha=0.1),
        Dropout(0.3),

        Dense(32),
        BatchNormalization(),
        tf.keras.layers.LeakyReLU(alpha=0.1),

        Dense(16, activation="relu"),
        Dense(1, activation="sigmoid")
    ])

    model.compile(optimizer=AdamW(learning_rate=0.001),
                  loss="binary_crossentropy", metrics=["accuracy"])
    return model

# Train the model
model = create_model()
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_loss",
                                                  patience=10, restore_best_weights=True)

model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test),
        batch_size=16, callbacks=[early_stopping])

# Save the model
model.save("diabetes_model2.h5")

```

Chi tiết cải tiến:

BatchNormalization()

Chuẩn hóa đầu vào của từng lớp về phân phối chuẩn (mean = 0, std = 1). Giúp mô hình hội tụ nhanh hơn, giảm hiện tượng "internal covariate shift". Hạn chế overfitting, đặc biệt là khi có nhiều tầng ẩn.

tf.keras.layers.LeakyReLU(alpha=0.1)

Một dạng ReLU cải tiến giúp tránh hiện tượng "dead neuron" khi đầu vào nhỏ hơn 0. Luôn cho ra giá trị khác 0 → mô hình học tốt hơn với dữ liệu phức tạp.

Dropout(0.4) # hoặc 0.3

Tắt ngẫu nhiên các node trong quá trình huấn luyện. Ngăn mô hình học quá khớp (overfitting). Ở đây dùng tỉ lệ dropout lớn hơn mô hình cũ (0.4 → mạnh hơn), phù hợp cho mô hình nhiều tầng.

Mô hình tổng quát:

Input → Dense → BatchNorm → LeakyReLU → Dropout → ... → Dense(sigmoid)

Dùng AdamW thay vì Adam

```
from tensorflow.keras.optimizers import AdamW
```

AdamW = Adam + Weight Decay (regularization trực tiếp lên trọng số).

Lợi ích:

- + Tránh overfitting tốt hơn.
- + Ổn định hơn với dữ liệu không đồng đều.
- + Tối ưu hóa giống như L2 Regularization một cách rõ ràng hơn Adam thường.

`EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)`

Theo dõi `val_loss`: nếu không giảm sau 10 lần (`patience=10`), sẽ dừng huấn luyện.
`restore_best_weights=True`: đảm bảo trọng số tốt nhất sẽ được lưu lại, tránh việc lấy model tại epoch cuối. Cách hiệu quả để ngăn overfitting, tiết kiệm thời gian huấn luyện.

Chuẩn hóa đầu vào (`StandardScaler`)

Chuẩn hóa toàn bộ dữ liệu đầu vào giúp: Các feature có cùng phân phối → mô hình hội tụ nhanh hơn. Tránh feature lớn hơn “áp đảo” các feature nhỏ hơn.

Step 2: Create a Flask API

```
from flask import Flask, request, jsonify
import numpy as np
import tensorflow as tf

app = Flask(__name__)

# Load trained model
model = tf.keras.models.load_model("diabetes_model2.h5")

@app.route("/predict", methods=["POST"])
def predict():
    data = request.json
    features = np.array(data["features"]).reshape(1, -1)
    prediction = model.predict(features)[0][0]
    result = "Diabetic" if prediction > 0.5 else "Non-Diabetic"

    return jsonify({"prediction": result, "probability": float(prediction)})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Step 3.1: Tạo Dockerfile để deploy trên Docker

```
# Sử dụng Python 3.9 slim image
FROM python:3.9-slim

# Thiết lập thư mục làm việc trong container
```

```
WORKDIR /app
```

```
# Copy các file cần thiết vào container
```

```
COPY app4.py diabetes_model2.h5 /app/
```

```
# Tạo requirements.txt để cài thư viện dễ dàng hơn
```

```
RUN echo "flask\ntensorflow-cpu\numpy" > requirements.txt
```

```
# Cài đặt các dependencies hệ thống cần thiết
```

```
RUN apt-get update && apt-get install -y gcc libglib2.0-0 libsm6 libxrender1  
libxext6
```

```
# Cài đặt các thư viện từ requirements.txt
```

```
RUN pip install --no-cache-dir --default-timeout=1000 -r requirements.txt
```

```
# Mở port 5000 để chạy API
```

```
EXPOSE 5000
```

```
# Chạy Flask app khi container khởi động
```

```
CMD ["python", "app4.py"]
```

Kiểm tra xem Docker có đang chạy hay không

```
(.venv) PS D:\BTL-AI\BT-TTCS> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8361334dbc5b	diabetes-api	"python app4.py"	3 minutes ago	Up 3 minutes	0.0.0.0:5000->5000/tcp	happy_mccarthy

Step 3.1.2 : Front-End

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title>Diabetes Predictor</title>
```

```
<style>
```

```
body {
```

```
font-family: Arial, sans-serif;
```

```
max-width: 600px;
```

```
margin: 50px auto;
```

```
padding: 10px;
```

```
border: 1px solid #ccc;
```

```
box-shadow: 5px 5px 5px rgba(165, 165, 165, 0.1);
```

```
border-radius: 12px;
```

```
background: #ffffff;
```

```
}
```

```
input {
```

```
width: 90%;
```

```
margin: 8px 0;
```

```
padding: 10px;
```

```
border-radius: 10px;
```

```
border: 1px solid #ccc;
```

```
}
```

```
button {
```

```
padding: 10px 20px;
```

```
margin-top: 10px;
```

```
background-color: #007bff;
```

```
color: white;
```

```
border: none;
```

```
border-radius: 6px;
```

```

        cursor: pointer;
    }
    .result {
        margin-top: 20px;
        font-weight: bold;
    }
</style>
</head>
<body>

<h2>☐ Dự đoán Tiểu Đường</h2>

<form id="predictForm">
    <label>Nhập 8 giá trị đặc trưng:</label>
    <input type="text" id="features" placeholder="VD:
6,148,72,35,0,33.6,0.627,50" required />
    <button type="submit">Dự đoán</button>
</form>

<div class="result" id="result"></div>

<script>
    document.getElementById('predictForm').addEventListener('submit', async
function(e) {
        e.preventDefault();

        const input = document.getElementById('features').value;
        const features = input.split(',').map(Number);

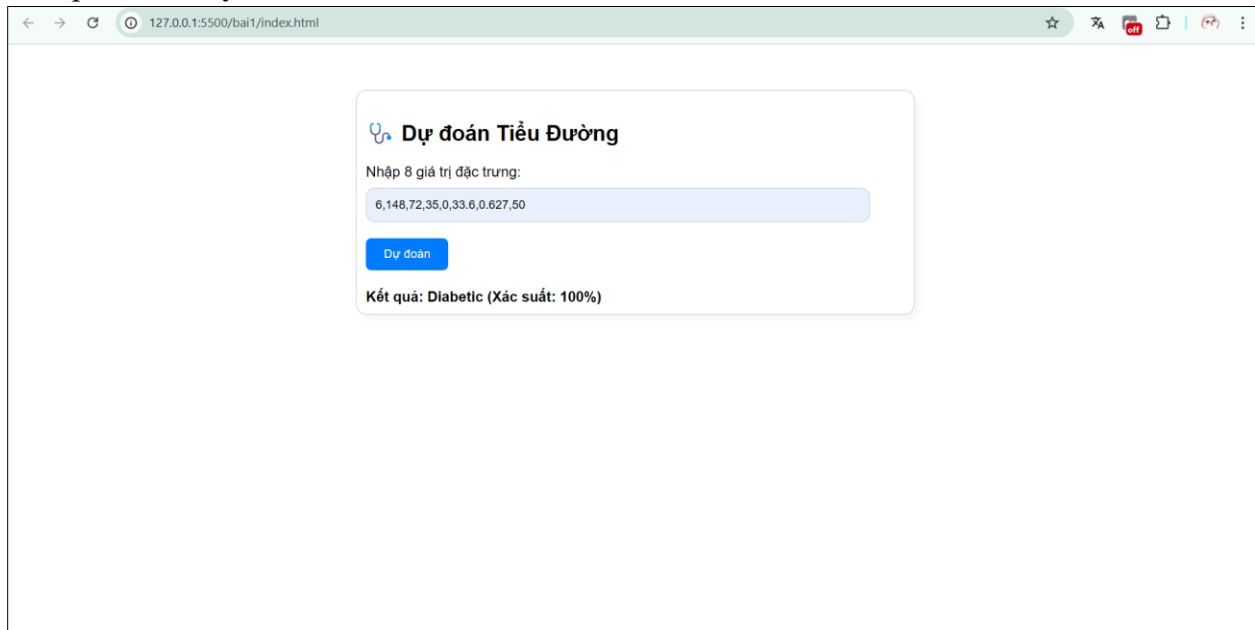
        const res = await fetch('http://localhost:5000/predict', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ features })
        });

        const data = await res.json();
        document.getElementById('result').textContent =
            `Kết quả: ${data.prediction} (Xác suất: ${Math.round(data.probability
* 100)}%)`;
    });
</script>

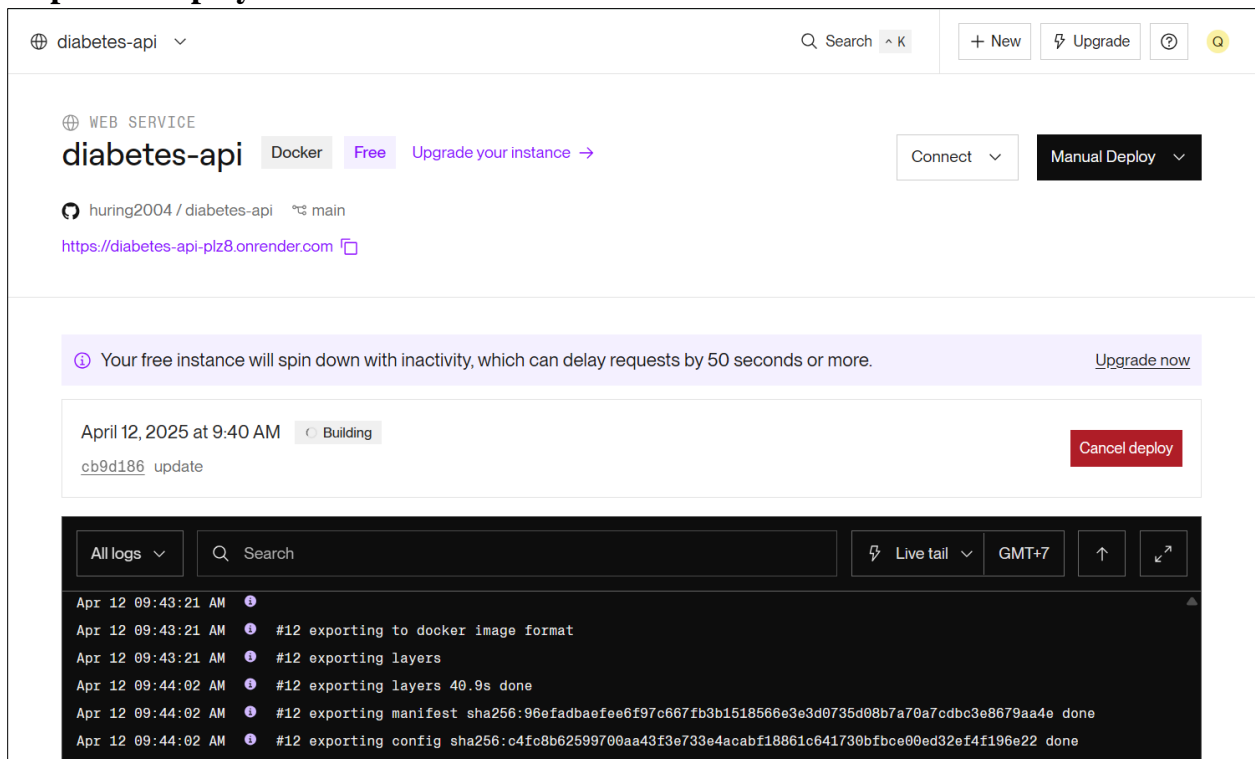
</body>
</html>

```


Kết quả khi chạy



Step 3.2 : Deploy trên Render



Step 3.2.1 : Test bằng Postman

gửi POST đến:

`https://diabetes-api-plz8.onrender.com/predict`

POST https://diabetes-api-pl

No environment

https://diabetes-api-plz8.onrender.com/predict

Save Share

POST

https://diabetes-api-plz8.onrender.com/predict

Send

Params

Authorization

Headers (10)

Body

Scripts

Settings

Cookies

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

Beautify

```
1 {"features": [6, 100, 72, 35, 0, 1000, 0.6, 50]}
```

Body

Cookies

Headers (12)

Test Results

200 OK

896 ms

414 B

🌐

⋮

{ } JSON

Preview

Visualize

⋮

```
1 {
2   "prediction": "Diabetic",
3   "probability": 1.0
4 }
```