Diabetes Detection Documentation using CNN and FASTAPI

🧠 File: `Diabetes_CNN_Detection.ipynb`

Notebook Purpose:

Train a Convolutional Neural Network to detect diabetes from tabular data.

1. Importing Libraries

```
python
CopyEdit

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv1D,
MaxPooling1D, Flatten
```

Explanation:

- `numpy`, `pandas`: For handling arrays and dataframes.
- `train_test_split`: Splits data into training and testing sets.
- `StandardScaler`: Normalizes the features.
- `tensorflow.keras`: Used for building and training the CNN.

2. Load the Dataset

```
python
CopyEdit
df = pd.read_csv('diabetes.csv')
df.head()
```

Explanation:

- Reads the diabetes dataset from a CSV file.
- Displays the first few rows to verify data format.

3. Splitting Features and Labels

```
python
CopyEdit
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

Explanation:

- X: Features (e.g., glucose level, insulin, BMI).
- y: Labels (0 = no diabetes, 1 = diabetes).

4. Preprocessing: Scaling the Features

```
python
CopyEdit
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Explanation:

- Normalizes features for better neural network convergence.

5. Reshape for CNN Input

```
python
CopyEdit
X_scaled = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)
```

Explanation:

- Reshapes the data to 3D format `(samples, features, channels)` required by Conv1D.

6. Split into Train/Test Sets

```
python
CopyEdit
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
```

Explanation:

- 80% training data, 20% testing.

7. Build the CNN Model

```
python
CopyEdit
model = Sequential()
model.add(Conv1D(32, 3, activation='relu',
input_shape=(X_scaled.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

Explanation:

- `Conv1D`: Detects patterns in feature sequences.
- `MaxPooling1D`: Downsamples features.
- `Flatten`: Converts to 1D before dense layers.
- `Dense`: Fully connected layers.
- `Dropout`: Prevents overfitting.
- `Sigmoid`: Output layer for binary classification.

8. Compile the Model

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

Explanation:

- Uses Adam optimizer.
- `binary_crossentropy`: Suitable for binary classification.
- Tracks accuracy during training.

9. Train the Model

```
python
CopyEdit
history = model.fit(X_train, y_train, epochs=30, batch_size=16,
validation_data=(X_test, y_test))
```

Explanation:

- Trains for 30 epochs with batch size of 16.
- Validates on test set.

10. Evaluate Performance

```
python
CopyEdit
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy}")
```

Explanation:

- Evaluates model on unseen data.
- Prints accuracy.

11. Save the Model

```
python
CopyEdit
model.save("diabetes_cnn_model.h5")
```

Explanation:

- Saves the trained model in HDF5 format for deployment.

🧪 File: `main.ipynb`

This notebook likely:

- Loads the saved model.
- Accepts or simulates new input.
- Performs prediction.

1. Import Required Libraries

```
python
CopyEdit
import numpy as np
from tensorflow.keras.models import load_model
import pandas as pd
from sklearn.preprocessing import StandardScaler
```

2. Load and Preprocess Test Input

```
python
CopyEdit
input_data = pd.read_csv("new_data.csv")
scaler = StandardScaler()
X_scaled = scaler.fit_transform(input_data)
X_scaled = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)
```

Explanation:

- Loads new data.
- Scales and reshapes like in training.

3. Load the Trained Model

```
python
CopyEdit
model = load_model("diabetes_cnn_model.h5")
```

4. Predict

```
python
CopyEdit
```

```
predictions = model.predict(X_scaled)
predictions = (predictions > 0.5).astype(int)
print(predictions)
```

Explanation:

- Generates predictions and thresholds them at 0.5.
- Outputs 0 or 1.

🧩 Project Integration

All these components work together:

1. **Training (`Diabetes_CNN_Detection.ipynb`)**: Trains and saves the model.
2. **Testing (`main.ipynb`)**: Uses the saved model for prediction.
3. **FastAPI App (not shown but implied as `app.main:app`)**: A web API to serve predictions.
4. **`start.sh`**: Starts the FastAPI app using `uvicorn`.
5. **`Dockerfile`**: Wraps everything in a container for easy deployment.