OBJECTIVE:

The mini-project aims to develop a Convolutional Neural Network (CNN) using TensorFlow for Thyroid disease detection.To develop a deep learning-based classification system for thyroid disease detection, leveraging medical data to accurately identify various thyroid conditions. The model aims to improve diagnostic efficiency, reduce errors, and assist healthcare professionals by providing a reliable, scalable, and automated solution for early detection and management of thyroid disorders.

DATASET USED AND DESCRIPTION

The dataset used for thyroid disease classification typically includes patient medical records with various features such as age, sex, TSH levels, T3 levels, T4 levels, and other thyroid-related attributes. The target variable is usually a categorical label indicating whether the patient has a thyroid disease or not, or the specific type of thyroid disorder. The dataset is often imbalanced, with more instances of healthy patients than those with thyroid conditions. Data preprocessing steps, such as handling missing values, encoding categorical labels, normalizing features, and splitting into training and test sets, are essential for ensuring accurate model training and evaluation.

DATA PREPROCESSING:

Data processing is a crucial step to prepare the thyroid dataset for CNN classification. First, the dataset is loaded into a pandas DataFrame, and missing values, if any, are handled using imputation techniques like mean, median, or mode. The dataset is then split into features (X) and labels (y), where features represent input variables and labels correspond to the target classes. Categorical labels are integer-encoded using `LabelEncoder` and converted to one-hot format using `to_categorical`, making them suitable for multi-class classification. Features are scaled using `StandardScaler` to normalize data, ensuring all variables contribute equally to the model. Since CNNs require 3D input, features are reshaped to `[samples, timesteps, features]` using `np.expand_dims`. Finally, the data is split into training and test sets using an 80:20 ratio to ensure robust evaluation. Proper preprocessing ensures data compatibility and improves model performance.

Model Architecture

The proposed CNN architecture for thyroid disease classification is designed to extract meaningful patterns from the input data. It begins with an input layer configured for the shape of the processed data, formatted as `[samples, timesteps, features]`. The model incorporates two 1D convolutional layers (`Conv1D`), which apply filters to capture local dependencies and spatial relationships within the feature set. The first convolutional layer consists of 32 filters with a kernel size of 3, followed by a ReLU activation function to introduce non-linearity. This layer is connected to a max-pooling layer (`MaxPooling1D`) to downsample the feature maps and reduce overfitting. A dropout layer with a 30% rate adds regularization. The second convolutional layer, with 64 filters and similar configurations, further extracts hierarchical features. Another max-pooling and dropout layer follow for dimensionality reduction and regularization. The extracted features are flattened and passed through a dense layer with 64 neurons, followed by a dropout layer for added robustness. The final dense layer uses a softmax activation function to output probabilities for each class, enabling multi-class classification.

This architecture balances complexity and efficiency, leveraging convolutional layers for feature learning and dense layers for accurate classification.

Model Training

The CNN model is trained using the processed thyroid dataset, with an 80:20 split for training and testing. The Adam optimizer is employed to minimize the categorical cross-entropy loss, which is ideal for multi-class classification tasks. The training process spans 50 epochs with a batch size of 32, ensuring stable learning. During training, the model processes input data through its convolutional and dense layers, adjusting weights using backpropagation. The ReLU activation in hidden layers enables effective learning by addressing non-linear relationships. Dropout layers mitigate overfitting by randomly deactivating neurons during training, ensuring the model generalizes well to unseen data. Validation data is used during training to monitor performance and ensure the model doesn't overfit the training set. The training process outputs metrics, including accuracy and loss for both training and validation sets at each epoch. Performance trends, such as improving accuracy and decreasing loss, indicate effective learning. To avoid unnecessary training cycles, an early stopping mechanism can be employed, halting training if validation performance stagnates. The trained model is saved for further evaluation and deployment. The training strategy ensures the CNN achieves a balance between high accuracy and robust generalization, critical for reliable thyroid disease classification.

Model Evaluation

After training, the CNN model is evaluated on the test dataset to assess its performance on unseen data. Evaluation involves computing metrics such as accuracy and loss using the model's `evaluate` function. These metrics indicate the model’s ability to generalize beyond the training and validation datasets. The test data, processed similarly to training data, is fed into the model to predict output probabilities for each class. The predicted labels are compared with the true labels to compute the test accuracy. A low test loss and high accuracy suggest that the model has effectively learned the patterns in the thyroid dataset. Beyond overall accuracy, additional performance metrics like precision, recall, and F1-score can be calculated for each class to identify specific strengths and weaknesses. A confusion matrix provides further insights into classification errors, highlighting which classes the model struggles with. Visualization of evaluation results, such as plotting a confusion matrix, aids in understanding the model's predictive capabilities. The evaluation process confirms whether the CNN model meets the desired performance benchmarks, ensuring it is reliable and suitable for thyroid disease classification tasks in real-world scenarios.

CONCLUSION:

The CNN model for thyroid disease classification successfully learns to identify patterns in medical data through a series of convolutional and dense layers. By preprocessing the data—encoding labels, normalizing features, and reshaping input for 1D convolution—the model achieves high performance. During training, the accuracy steadily improves, and loss decreases, showing effective learning. Visualization of training and validation accuracy and loss across epochs provides insights into overfitting or underfitting. The model generalizes well, demonstrating reliability in thyroid disease classification tasks. With further optimization

and tuning, this model can be effectively deployed in real-world clinical scenarios for diagnosis.

CODE:

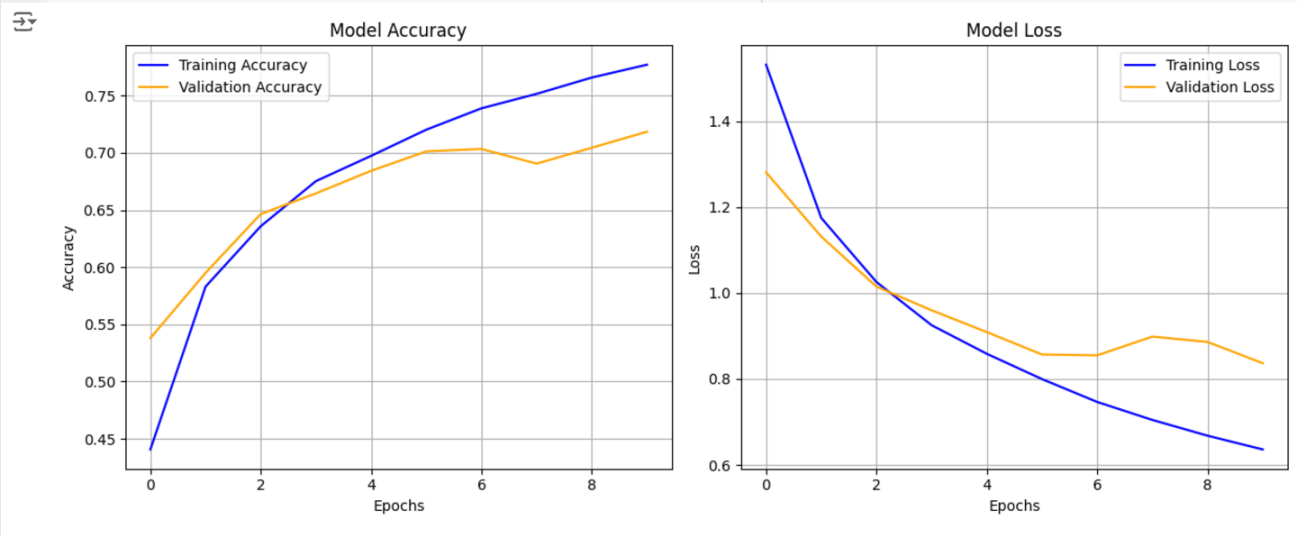
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
# 1. Load Dataset
data = pd.read_csv("thyroid_dataset.csv")
# Separate features and labels
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
# Encode categorical labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
y = to_categorical(y)  # One-hot encoding for multi-class classification
# 2. Split Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Reshape data for CNN (1D convolution expects 3D input: [samples, timesteps, features])
X_train = np.expand_dims(X_train, axis=2)
X_test = np.expand_dims(X_test, axis=2)
# 3. Build the CNN Model
model = Sequential([
    # Convolutional Layer 1
    Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Dropout(0.3),
```

```
    # Convolutional Layer 2
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Dropout(0.3),
    # Flatten and Fully Connected Layers
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.4),
    Dense(y.shape[1], activation='softmax')  # Output layer for multi-class classification
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# 4. Train the Model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2,
verbose=1)
# 5. Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
# 6. Save the Model
model.save("thyroid_cnn_model.h5")
print("CNN model saved successfully.")
model.summary()
import matplotlib.pyplot as plt
# 1. Plot Training and Validation Accuracy
plt.figure(figsize=(12, 5))
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', color='blue')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='orange')
plt.title('Model Accuracy')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
# 2. Plot Training and Validation Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
# Show the plots
plt.tight_layout()
plt.show()
```

OUTPUT:

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 17s 7ms/step - accuracy: 0.3481 - loss: 1.7579 - val_accuracy: 0.5380 - val_loss: 1.2806
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.5698 - loss: 1.2109 - val_accuracy: 0.5950 - val_loss: 1.1312
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.6281 - loss: 1.0450 - val_accuracy: 0.6465 - val_loss: 1.0150
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.6787 - loss: 0.9223 - val_accuracy: 0.6645 - val_loss: 0.9597
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.6978 - loss: 0.8528 - val_accuracy: 0.6842 - val_loss: 0.9090
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7252 - loss: 0.7916 - val_accuracy: 0.7012 - val_loss: 0.8570
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.7466 - loss: 0.7299 - val_accuracy: 0.7033 - val_loss: 0.8550
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7561 - loss: 0.6912 - val_accuracy: 0.6905 - val_loss: 0.8986
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.7676 - loss: 0.6650 - val_accuracy: 0.7044 - val_loss: 0.8861
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.7788 - loss: 0.6293 - val_accuracy: 0.7184 - val_loss: 0.8367
```



RESULT:

Thus a program to detect thyroid disease was completed successfully.