Python – Commonly Used Library and Functions with examples

AI, Neural Networks.

Contents

[Python Library Functions Overview 4](#_Toc178835817)

[1. Standard Python Libraries 4](#_Toc178835818)

[1.1 os 4](#_Toc178835819)

[1.2 sys 4](#_Toc178835820)

[1.3 math 5](#_Toc178835821)

[1.4 json 5](#_Toc178835822)

[2. NumPy 6](#_Toc178835823)

[3. Plotly 6](#_Toc178835824)

[4. Seaborn 6](#_Toc178835825)

[5. Scikit-learn (sklearn) 7](#_Toc178835826)

[6. googlesearch-python 7](#_Toc178835827)

[7. BeautifulSoup (bs4) 8](#_Toc178835828)

[8. Pillow (PIL) 8](#_Toc178835829)

[9. Keras 8](#_Toc178835830)

[AI Program with Python 10](#_Toc178835831)

[1. Importing Libraries 10](#_Toc178835832)

[3. Data Visualization 10](#_Toc178835833)

[4. Data Preprocessing 11](#_Toc178835834)

[Code 13](#_Toc178835835)

[Deep Learning Program: Image Classification with Convolutional Neural Networks (CNN) 15](#_Toc178835836)

[Overview 15](#_Toc178835837)

[How the Program Works 15](#_Toc178835838)

[Flow Chart 16](#_Toc178835839)

[Loading the Dataset: The CIFAR-10 dataset is loaded using Keras, which automatically downloads and splits the data into training and testing sets. 17](#_Toc178835840)

[**Preprocessing the Data**: The images are normalized to have pixel values between 0 and 1. This step helps in faster convergence of the model during training. 17](#_Toc178835841)

[**Defining the Model**: A CNN model is defined using Keras. The architecture consists of several layers: 17](#_Toc178835842)

[Compiling the Model: The model is compiled with an optimizer, loss function, and evaluation metric: 17](#_Toc178835843)

[**Training the Model**: The model is trained on the training dataset. The training process involves adjusting the weights based on the loss computed from the predictions and the actual labels. 18](#_Toc178835844)

[**Evaluating the Model**: After training, the model is evaluated on the test dataset to assess its performance. The accuracy and loss are reported. 18](#_Toc178835845)

[Making Predictions: 18](#_Toc178835846)

[Explanation of Techniques Used 18](#_Toc178835847)

[Full Program for Deep Learning 19](#_Toc178835848)

[Test Accuracy 21](#_Toc178835849)

[Significance 21](#_Toc178835850)

[Limitations 21](#_Toc178835851)

[Neural networks 22](#_Toc178835852)

[Flow Chart 22](#_Toc178835853)

[Step By Step Flow 24](#_Toc178835854)

[Importing Libraries: 24](#_Toc178835855)

[Loading the Dataset: 24](#_Toc178835856)

[Data Normalization: 24](#_Toc178835857)

[One-Hot Encoding of Labels: 24](#_Toc178835858)

[Building the Neural Network Model: 24](#_Toc178835859)

[Compiling the Model: 24](#_Toc178835860)

[Training the Model: 24](#_Toc178835861)

[Evaluating the Model: 24](#_Toc178835862)

[Visualizing Predictions: 24](#_Toc178835863)

[Code 26](#_Toc178835864)

# Python Library Functions Overview

## 1. Standard Python Libraries

### 1.1 os

The os module allows interaction with the operating system, enabling file and directory manipulation.

Common Functions:

* os.listdir(path) – Lists files and directories in the given path.
* os.path.exists(path) – Checks if the given path exists.
* os.mkdir(path) – Creates a directory at the specified path.
* os.remove(path) – Removes a file at the specified path.

#### Example:

import os  
  
# List files in the current directory  
files = os.listdir('.')  
print(files)  
  
# Create a new directory  
os.mkdir('new\_folder')

### 1.2 sys

The sys module provides access to system-specific parameters and functions.

Common Functions:

* sys.exit() – Exits from Python.
* sys.argv – A list of command-line arguments passed to a script.
* sys.version – Returns the version of the Python interpreter.

#### Example:

import sys  
  
# Get command-line arguments  
print(sys.argv)  
  
# Exit the program  
if len(sys.argv) < 2:  
 print("Usage: python script.py <argument>")  
 sys.exit()

### 1.3 math

The math module provides mathematical functions for trigonometry, logarithms, and more.

Common Functions:

* math.sqrt(x) – Returns the square root of x.
* math.pi – Returns the value of π.
* math.factorial(n) – Returns the factorial of n.
* math.sin(x) – Returns the sine of x (in radians).

#### Example:

import math  
  
# Calculate the square root  
result = math.sqrt(16)  
print(result) # Output: 4.0  
  
# Calculate the sine of pi/2  
sine\_value = math.sin(math.pi / 2)  
print(sine\_value) # Output: 1.0

### 1.4 json

The json module helps in parsing JSON data, allowing easy conversion between Python objects and JSON format.

Common Functions:

* json.load(file) – Parses JSON from a file.
* json.dumps(obj) – Converts a Python object to a JSON string.
* json.loads(json\_string) – Parses a JSON string into a Python object.

#### Example:

import json  
  
# Convert Python object to JSON  
data = {'name': 'John', 'age': 30}  
json\_data = json.dumps(data)  
print(json\_data)  
  
# Load JSON from a file  
with open('data.json', 'r') as file:  
 loaded\_data = json.load(file)  
 print(loaded\_data)

## 2. NumPy

The numpy library is essential for numerical computing in Python, focusing on array and matrix operations.

Common Functions:

* numpy.array() – Creates an array.
* numpy.dot() – Computes the dot product of two arrays.
* numpy.mean() – Returns the mean of array elements.
* numpy.arange() – Creates an array with a range of numbers.

#### Example:

import numpy as np  
  
# Create an array  
arr = np.array([1, 2, 3, 4])  
print(arr.mean()) # Output: 2.5  
  
# Calculate dot product  
arr1 = np.array([1, 2])  
arr2 = np.array([3, 4])  
dot\_product = np.dot(arr1, arr2)  
print(dot\_product) # Output: 11

## 3. Plotly

plotly is a graphing library that creates interactive and publication-quality graphs.

Common Functions:

* plotly.graph\_objs.Figure() – Creates a figure for plotting.
* plotly.express.line() – Creates a line chart.
* plotly.express.scatter() – Creates a scatter plot.

#### Example:

import plotly.express as px  
  
# Line chart example  
data = [10, 20, 30, 40]  
fig = px.line(data, title="Simple Line Chart")  
fig.show()

## 4. Seaborn

Seaborn is a visualization library based on Matplotlib for creating informative and attractive statistical graphics.

Common Functions:

* seaborn.heatmap() – Creates a heatmap.
* seaborn.pairplot() – Creates a pairwise plot of variables.
* seaborn.boxplot() – Creates a box plot.

#### Example:

import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Heatmap example  
data = [[1, 2], [3, 4]]  
sns.heatmap(data, annot=True)  
plt.show()

## 5. Scikit-learn (sklearn)

Scikit-learn is a powerful machine learning library in Python that provides various algorithms and utilities for data analysis and modeling.

Common Functions:

* sklearn.linear\_model.LinearRegression() – Implements linear regression.
* sklearn.model\_selection.train\_test\_split() – Splits data into training and testing sets.
* sklearn.preprocessing.StandardScaler() – Standardizes features.

#### Example:

from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
  
# Simple Linear Regression example  
X = [[1], [2], [3]]  
y = [1, 2, 3]  
model = LinearRegression()  
model.fit(X, y)  
print(model.coef\_) # Output: [1.]

## 6. googlesearch-python

googlesearch-python allows you to perform Google searches programmatically.

Common Functions:

* search(query, num\_results) – Performs a Google search with the given query.

#### Example:

from googlesearch import search  
  
# Perform a search  
for result in search('Python programming', num\_results=5):  
 print(result)

## 7. BeautifulSoup (bs4)

BeautifulSoup is used for parsing HTML and XML documents to extract data easily.

Common Functions:

* BeautifulSoup() – Parses an HTML document.
* find\_all() – Finds all matching tags in the document.
* find() – Finds the first matching tag.

#### Example:

from bs4 import BeautifulSoup  
  
# Parse HTML  
html = "<html><head><title>Page Title</title></head><body><p>Paragraph</p></body></html>"  
soup = BeautifulSoup(html, 'html.parser')  
print(soup.title.string) # Output: Page Title

## 8. Pillow (PIL)

Pillow is an image processing library in Python for opening, manipulating, and saving image files.

Common Functions:

* Image.open() – Opens an image file.
* Image.save() – Saves an image to a file.
* Image.show() – Displays an image.

#### Example:

from PIL import Image  
  
# Open and display an image  
img = Image.open('example.jpg')  
img.show()

## 9. Keras

Keras is a high-level deep learning library that runs on top of TensorFlow, simplifying the process of building and training neural networks.

Common Functions:

* keras.models.Sequential() – Creates a neural network model.
* keras.layers.Dense() – Adds a dense (fully connected) layer to a model.
* keras.models.compile() – Configures the model for training.

#### Example:

from keras.models import Sequential  
from keras.layers import Dense  
  
# Simple neural network model  
model = Sequential()  
model.add(Dense(32, input\_dim=10, activation='relu'))  
model.compile(loss='binary\_crossentropy', optimizer='adam')

# AI Program with Python

**Overview**

This program predicts housing prices based on various features of houses using a linear regression model. It leverages data from the Boston Housing dataset, which includes multiple attributes such as crime rate, number of rooms, and proximity to the Charles River, among others. The program is structured to provide an interactive exploration of the dataset, perform statistical analysis, and visualize the results through distinct graphs for better understanding.

**How the Program Works**

1. Importing Libraries: The program begins by importing necessary libraries:
   * **Pandas**: For data manipulation and analysis.
   * **NumPy**: For numerical operations.
   * **Matplotlib** and **Seaborn**: For data visualization.
   * **Scikit-learn**: For machine learning, specifically for splitting the dataset and building the linear regression model.
2. **Loading the Dataset**: The program loads the Boston Housing dataset directly from a URL. It assigns descriptive column names to enhance readability.

python

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv"

columns = ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT", "MEDV"]

data = pd.read\_csv(url, header=None, names=columns)

1. Data Visualization: To understand the distribution and relationships within the data, the program includes various visualizations:
   * **Histogram**: Displays the distribution of house prices (MEDV) to understand how prices are spread.
   * **Scatter Plot**: Shows the relationship between the average number of rooms (RM) and house prices, allowing users to see how room count affects pricing.
   * **Boxplot**: Compares house prices based on proximity to the Charles River (CHAS), illustrating the impact of location on pricing.

Each visualization is tailored with specific styles and colors to make them distinct and informative.

1. Data Preprocessing: The dataset is split into features (X) and the target variable (y), which represents the house prices. The data is then divided into training and testing sets (80% training and 20% testing) to facilitate model training and evaluation.

python

X = data.drop("MEDV", axis=1) # Features

y = data["MEDV"] # Target variable (house price)

1. **Training the Model**: A linear regression model is instantiated and trained on the training dataset. This model learns the relationships between the features and the target variable.

python

model = LinearRegression()

model.fit(X\_train, y\_train)

1. **Making Predictions**: After training, the model predicts housing prices on the test dataset. These predictions are then compared with actual prices to evaluate the model’s performance.
2. **Evaluation Metrics**: The program calculates the Mean Squared Error (MSE) and R-squared score to assess the model's accuracy:
   * **Mean Squared Error (MSE)**: Measures the average squared difference between predicted and actual values. A lower MSE indicates better prediction accuracy.
   * **R-squared Score**: Indicates the proportion of variance in the dependent variable (house prices) that can be explained by the independent variables (features). An R-squared close to 1 suggests a good fit.
3. **Visualizing Results**:
   * A scatter plot of actual vs. predicted prices is generated, providing a visual indication of prediction accuracy.
   * A residuals plot is included to help identify patterns in the prediction errors, which can indicate whether the model is appropriately capturing the underlying relationships.
4. **Feature Importance**: The program calculates the importance of each feature in predicting house prices. This helps identify which features have the most significant influence, providing insights into what factors affect housing prices the most.

# Code

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Set up visual styles

sns.set(style="whitegrid")

# Step 1: Load the dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv"

columns = ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT", "MEDV"]

data = pd.read\_csv(url, header=None, names=columns)

# Display the first few rows of the dataset

print("Dataset Preview:")

print(data.head())

# Step 2: Data Visualization

# 2.1 Histogram for House Prices (MEDV)

plt.figure(figsize=(10, 6))

sns.histplot(data['MEDV'], bins=30, kde=True, color='blue', edgecolor='black')

plt.title('Distribution of House Prices (MEDV)', fontsize=16)

plt.xlabel('Price', fontsize=14)

plt.ylabel('Frequency', fontsize=14)

plt.grid(axis='y')

plt.show()

# 2.2 Scatter plot to visualize relationships

plt.figure(figsize=(10, 6))

plt.scatter(data['RM'], data['MEDV'], color='orange', alpha=0.6, edgecolors='black')

plt.title('House Prices vs Number of Rooms', fontsize=16)

plt.xlabel('Average Number of Rooms (RM)', fontsize=14)

plt.ylabel('House Prices (MEDV)', fontsize=14)

plt.grid()

plt.show()

# 2.3 Boxplot for feature comparison

plt.figure(figsize=(12, 6))

sns.boxplot(x='CHAS', y='MEDV', data=data, palette='Set2')

plt.title('House Prices by Charles River Dummy Variable (CHAS)', fontsize=16)

plt.xlabel('Charles River Dummy Variable (0 = No, 1 = Yes)', fontsize=14)

plt.ylabel('House Prices (MEDV)', fontsize=14)

plt.grid(axis='y')

plt.show()

# Step 3: Preprocess the data

X = data.drop("MEDV", axis=1) # Features

y = data["MEDV"] # Target variable (house price)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Display the training and testing data shapes

print(f"Training Data Shape: {X\_train.shape}")

print(f"Testing Data Shape: {X\_test.shape}")

# Step 4: Train the model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Step 5: Make predictions

y\_pred = model.predict(X\_test)

# Step 6: Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

# Calculate and display the R-squared score

r\_squared = model.score(X\_test, y\_test)

print(f'R-squared Score: {r\_squared:.2f}')

# Human-readable interpretation of MSE and R-squared

print(f'\nThe Mean Squared Error (MSE) of {mse:.2f} indicates the average squared difference between predicted and actual prices. A lower MSE means better predictions.')

print(f'The R-squared score of {r\_squared:.2f} means that about {r\_squared \* 100:.1f}% of the variance in house prices can be explained by the features used in this model.')

# Visualization: Actual vs Predicted Prices

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred, color='purple', alpha=0.6, edgecolors='black')

plt.xlabel('Actual Prices', fontsize=14)

plt.ylabel('Predicted Prices', fontsize=14)

plt.title('Actual vs Predicted House Prices', fontsize=16)

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red', linestyle='--', linewidth=2)

plt.grid()

plt.show()

# Step 7: Residuals plot

plt.figure(figsize=(10, 6))

sns.residplot(x=y\_pred, y=y\_test - y\_pred, lowess=True, color="g", line\_kws={'color': 'red', 'lw': 2})

plt.xlabel('Predicted Prices', fontsize=14)

plt.ylabel('Residuals', fontsize=14)

plt.title('Residuals vs Predicted Prices', fontsize=16)

plt.axhline(0, color='red', linestyle='--', linewidth=2)

plt.grid()

plt.show()

# Step 8: Provide a summary of the important features

feature\_importance = pd.Series(model.coef\_, index=X.columns).sort\_values(ascending=False)

print("\nFeature Importance (Influence on House Price):")

print(feature\_importance)

# Human-readable interpretation of important features

print("\nThe following features have the most influence on house prices (MEDV):")

for feature, importance in feature\_importance.items():

print(f"{feature}: {'Higher' if importance > 0 else 'Lower'} values are associated with {'higher' if importance > 0 else 'lower'} house prices.")

# Deep Learning Program: Image Classification with Convolutional Neural Networks (CNN)

# Overview

This program is designed to classify images from the CIFAR-10 dataset, which contains 60,000 32x32 color images across 10 different classes (e.g., airplanes, cars, birds, etc.). The model uses a Convolutional Neural Network (CNN), a popular deep learning architecture for image processing tasks.

# How the Program Works

1. **Importing Libraries**: The program begins by importing necessary libraries:
   * **TensorFlow/Keras**: For building and training the neural network.
   * **NumPy**: For numerical operations.
   * **Matplotlib**: For data visualization.

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import matplotlib.pyplot as plt

## Flow Chart

Here's a simple text flow chart outlining the program's process:

Start

|

V

Import Libraries

|

V

Load CIFAR-10 Dataset

|

V

Preprocess Data (Normalization)

|

V

Define CNN Model

|-- Conv2D Layer 1 (32 filters, ReLU)

|-- MaxPooling2D Layer 1

|-- Conv2D Layer 2 (64 filters, ReLU)

|-- MaxPooling2D Layer 2

|-- Conv2D Layer 3 (128 filters, ReLU)

|-- MaxPooling2D Layer 3

|-- Flatten Layer

|-- Dense Layer (128 units, ReLU)

|-- Dense Output Layer (10 classes, Softmax)

|

V

Compile Model (Optimizer, Loss, Metrics)

|

V

Train Model (Fit to Training Data)

|

V

Evaluate Model (Test Accuracy)

|

V

Make Predictions

|

V

Display Results (Predicted Classes)

|

V

End

## Loading the Dataset: The CIFAR-10 dataset is loaded using Keras, which automatically downloads and splits the data into training and testing sets.

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.cifar10.load\_data()

## **Preprocessing the Data**: The images are normalized to have pixel values between 0 and 1. This step helps in faster convergence of the model during training.

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

## **Defining the Model**: A CNN model is defined using Keras. The architecture consists of several layers:

* + **Conv2D**: Convolutional layers that apply filters to the input image, allowing the model to learn spatial hierarchies and features.
  + **MaxPooling2D**: Reduces the dimensionality of the feature maps while retaining important features, helping to reduce overfitting and computation.
  + **Flatten**: Converts the 2D feature maps into a 1D vector for input into the fully connected layers.
  + **Dense**: Fully connected layers that output the final class probabilities.

model = keras.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Conv2D(128, (3, 3), activation='relu'),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(10, activation='softmax')

])

## Compiling the Model: The model is compiled with an optimizer, loss function, and evaluation metric:

* + **Adam**: An adaptive learning rate optimization algorithm that is efficient for large datasets.
  + **SparseCategoricalCrossentropy**: Used for multi-class classification tasks, comparing the predicted class probabilities with the true labels.
  + **Accuracy**: The metric used to evaluate the model's performance.

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

## **Training the Model**: The model is trained on the training dataset. The training process involves adjusting the weights based on the loss computed from the predictions and the actual labels.

model.fit(x\_train, y\_train, epochs=10, validation\_split=0.2)

## **Evaluating the Model**: After training, the model is evaluated on the test dataset to assess its performance. The accuracy and loss are reported.

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_acc}')

Making Predictions: The model can make predictions on new images. The predicted classes are then displayed alongside the actual images.

predictions = model.predict(x\_test)

predicted\_classes = np.argmax(predictions, axis=1)

# Display a few predictions

plt.figure(figsize=(10, 10))

for i in range(9):

plt.subplot(3, 3, i + 1)

plt.imshow(x\_test[i])

plt.title(f'Predicted: {predicted\_classes[i]}')

plt.axis('off')

plt.show()

## Explanation of Techniques Used

* **Convolutional Neural Network (CNN)**: A specialized deep learning architecture that excels in image recognition tasks. It uses convolutional layers to extract features from images, pooling layers to reduce dimensionality, and dense layers for classification.
* **Activation Function (ReLU)**: The Rectified Linear Unit (ReLU) activation function introduces non-linearity into the model, allowing it to learn complex patterns.
* **Max Pooling**: This technique reduces the spatial dimensions of the feature maps, which helps in reducing computational load and preventing overfitting.
* **Softmax Activation**: Used in the output layer for multi-class classification problems. It converts raw logits (output scores) into probabilities that sum to one.
* **Cross-Entropy Loss**: This loss function measures the difference between the predicted probabilities and the actual labels, guiding the optimization process.

# Full Program for Deep Learning

**import numpy as np**

**import pickle**

**import os**

**import matplotlib.pyplot as plt**

**import tensorflow as tf**

**from tensorflow import keras**

**from tensorflow.keras import layers**

**def load\_cifar10\_data(data\_dir='cifar-10-batches-py'):**

**print("Loading CIFAR-10 dataset...")**

**x\_train = []**

**y\_train = []**

**# Load training batches**

**for i in range(1, 6):**

**with open(os.path.join(data\_dir, f'data\_batch\_{i}'), 'rb') as f:**

**print(f"Processing data\_batch\_{i}...")**

**batch = pickle.load(f, encoding='bytes')**

**x\_train.append(batch[b'data']) # Use b'data'**

**y\_train.append(batch[b'labels'])**

**x\_train = np.concatenate(x\_train)**

**y\_train = np.concatenate(y\_train)**

**# Load test batch**

**with open(os.path.join(data\_dir, 'test\_batch'), 'rb') as f:**

**print("Processing test\_batch...")**

**test\_batch = pickle.load(f, encoding='bytes')**

**x\_test = test\_batch[b'data'] # Use b'data'**

**y\_test = test\_batch[b'labels']**

**# Reshape and normalize the data**

**x\_train = x\_train.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1) / 255.0**

**x\_test = x\_test.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1) / 255.0**

**print("Dataset loaded successfully.")**

**return (x\_train, y\_train), (x\_test, y\_test)**

**def build\_model(input\_shape):**

**print("Building the model...")**

**model = keras.Sequential([**

**layers.Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape),**

**layers.MaxPooling2D((2, 2)),**

**layers.Conv2D(64, (3, 3), activation='relu'),**

**layers.MaxPooling2D((2, 2)),**

**layers.Conv2D(64, (3, 3), activation='relu'),**

**layers.Flatten(),**

**layers.Dense(64, activation='relu'),**

**layers.Dense(10, activation='softmax')**

**])**

**model.compile(optimizer='adam',**

**loss='sparse\_categorical\_crossentropy',**

**metrics=['accuracy'])**

**print("Model built successfully.")**

**return model**

**def display\_sample\_images(x, y):**

**print("Displaying sample images...")**

**plt.figure(figsize=(10, 10))**

**for i in range(9):**

**plt.subplot(3, 3, i + 1)**

**plt.imshow(x[i])**

**plt.title(f'Label: {y[i]}')**

**plt.axis('off')**

**plt.show()**

**def main():**

**# Load the CIFAR-10 dataset**

**(x\_train, y\_train), (x\_test, y\_test) = load\_cifar10\_data()**

**# Display sample images from the training set**

**display\_sample\_images(x\_train, y\_train)**

**# Build the model**

**model = build\_model(input\_shape=(32, 32, 3))**

**# Train the model**

**print("Training the model...")**

**model.fit(x\_train, y\_train, epochs=10, validation\_data=(x\_test, y\_test))**

**# Evaluate the model**

**test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)**

**print(f'\nTest accuracy: {test\_acc}')**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

# Test Accuracy

Test accuracy is a critical metric in evaluating the performance of a machine learning model, particularly in classification tasks. Here's a detailed explanation of what it means:

Definition

Test accuracy refers to the proportion of correct predictions made by the model on a separate test dataset that it has not seen during training. It is expressed as a percentage and calculated using the formula:

A close up of a number

Description automatically generated

## Significance

1. Performance Evaluation:
   * Test accuracy provides a quantitative measure of how well the model generalizes to unseen data. A higher test accuracy indicates that the model has learned to recognize patterns in the data effectively.
2. Model Comparison:
   * It allows for the comparison of different models or algorithms. By evaluating their test accuracies, you can determine which model performs best on the specific classification task.
3. Overfitting vs. Underfitting:
   * Test accuracy can help identify issues such as overfitting and underfitting:
     + Overfitting: If a model has high training accuracy but low test accuracy, it may be overfitting the training data, meaning it has learned the noise or details in the training set rather than the underlying pattern.
     + Underfitting: If both training and test accuracies are low, the model is likely underfitting, indicating it has not learned enough from the training data to make good predictions.
4. Real-World Application:
   * In practical scenarios, test accuracy can provide insights into how well the model will perform when deployed in real-world applications. For example, in an image classification system, a high test accuracy would mean that the system is likely to classify images correctly when new images are input.

## Limitations

While test accuracy is an important metric, it should not be the only measure of model performance. It is also essential to consider:

* Precision: The accuracy of the positive predictions.
* Recall (Sensitivity): The ability of the model to find all the relevant cases (true positives).
* F1 Score: The harmonic mean of precision and recall, useful for imbalanced datasets.
* Confusion Matrix: A detailed breakdown of correct and incorrect predictions across different classes.

Summary

In summary, test accuracy is a key indicator of a model's ability to generalize to new, unseen data, and it plays a vital role in evaluating the overall effectiveness of a machine learning model in classification tasks.

# Neural networks

Here we will build a simple neural network using TensorFlow and Keras to classify handwritten digits from the MNIST dataset. This example illustrates how neural networks are built, trained, and evaluated, as well as their importance in image classification tasks.

**What You Will Learn**

1. **Building a Neural Network**: The code shows how to create a neural network model using Keras.
2. **Training the Model**: You'll see how the model is trained on a dataset.
3. **Evaluating Performance**: The code evaluates the model's performance on a test dataset.
4. **Visualizing Results**: The program includes visualizations of the predictions.

# Flow Chart

Start

|

v

+----------------------------+

| Import Libraries |

| (numpy, matplotlib, |

| tensorflow.keras) |

+----------------------------+

|

v

+----------------------------+

| Load the MNIST Dataset |

| (x\_train, y\_train), |

| (x\_test, y\_test) |

+----------------------------+

|

v

+----------------------------+

| Normalize Pixel Values |

| (0-255 to 0-1) |

+----------------------------+

|

v

+----------------------------+

| Convert Labels to Categorical|

| (One-hot encoding) |

+----------------------------+

|

v

+----------------------------+

| Build the Neural Model |

| (Sequential Model) |

| - Flatten Layer |

| - Dense Layer (128 units)|

| - Output Layer (10 units)|

+----------------------------+

|

v

+----------------------------+

| Compile the Model |

| (Optimizer: Adam, Loss: |

| Categorical Crossentropy)|

+----------------------------+

|

v

+----------------------------+

| Train the Model |

| (Fit to training data, |

| epochs=5, batch\_size=32, |

| validation\_split=0.2) |

+----------------------------+

|

v

+----------------------------+

| Evaluate the Model |

| (Test Accuracy) |

+----------------------------+

|

v

+----------------------------+

| Make Predictions |

| (Using test dataset) |

+----------------------------+

|

v

+----------------------------+

| Visualize Predictions |

| (Plot true vs predicted) |

+----------------------------+

|

v

End

# Step By Step Flow

## Importing Libraries:

* **Numpy**: Used for numerical operations, especially for handling arrays.
* **Matplotlib**: Used for visualizing images and results.
* **TensorFlow/Keras**: Used for building and training the neural network.

## Loading the Dataset:

* The MNIST dataset is loaded, which consists of training and testing sets of images and their corresponding labels.

## Data Normalization:

* The pixel values are normalized to a range of 0 to 1 by dividing by 255. This scaling helps the neural network learn more effectively.

## One-Hot Encoding of Labels:

* Labels are converted into a one-hot encoded format, making it easier for the model to predict multiple classes.

## Building the Neural Network Model:

* A sequential model is created, allowing layers to be added in a linear fashion.
* The input layer flattens the 28x28 images into a single vector.
* A hidden layer with 128 neurons uses the ReLU activation function to introduce non-linearity.
* The output layer has 10 neurons with a softmax activation function to output probabilities for each class.

## Compiling the Model:

* The model is compiled using the Adam optimizer, which is efficient for large datasets and categorical crossentropy loss function, suitable for multi-class problems.

## Training the Model:

* The model is trained on the training data for 5 epochs, with a batch size of 32.
* A portion of the training data (20%) is used for validation to monitor performance.

## Evaluating the Model:

* The model's performance is evaluated on the test dataset, and the accuracy is printed.

## Visualizing Predictions:

* A function is defined to visualize the model's predictions compared to the true labels.
* It plots the first 25 test images, showing both the true label and the predicted label.

# Code

# Import necessary libraries

import numpy as np # For numerical operations

import matplotlib.pyplot as plt # For plotting images

from tensorflow.keras.datasets import mnist # For loading the MNIST dataset

from tensorflow.keras.models import Sequential # For building a sequential model

from tensorflow.keras.layers import Dense, Flatten # For adding layers to the model

from tensorflow.keras.utils import to\_categorical # For converting labels to categorical format

# Load the MNIST dataset

# The dataset contains images of handwritten digits (0-9)

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Normalize the pixel values to be between 0 and 1

# This helps in faster convergence during training

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# Convert labels to categorical one-hot encoding

# Each label is converted to a binary vector of size 10

# For example, if the label is 3, it becomes [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

y\_train = to\_categorical(y\_train, num\_classes=10)

y\_test = to\_categorical(y\_test, num\_classes=10)

# Build the neural network model

model = Sequential([

Flatten(input\_shape=(28, 28)), # Flatten the 28x28 images into a 784-dimensional vector

Dense(128, activation='relu'), # Hidden layer with 128 neurons and ReLU activation function

Dense(10, activation='softmax') # Output layer with 10 neurons (one for each class) and softmax activation

])

# Compile the model

# Using Adam optimizer and categorical crossentropy loss function for multi-class classification

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

# Fit the model to the training data for 5 epochs

# Validation split is set to 0.2, meaning 20% of the training data will be used for validation

history = model.fit(x\_train, y\_train, epochs=5, batch\_size=32, validation\_split=0.2)

# Evaluate the model

# Evaluate the model's performance on the test dataset

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

print(f'Test Accuracy: {test\_accuracy:.4f}') # Print the test accuracy

# Define a function to visualize predictions

def plot\_predictions(images, true\_labels, predictions):

plt.figure(figsize=(12, 12)) # Set the figure size

for i in range(25): # Loop through the first 25 images

plt.subplot(5, 5, i + 1) # Create a subplot for each image

plt.imshow(images[i], cmap='gray') # Show the image in grayscale

plt.title(f'True: {np.argmax(true\_labels[i])}, Pred: {np.argmax(predictions[i])}') # Show true and predicted labels

plt.axis('off') # Turn off the axis

plt.show() # Display the plot

# Make predictions on the test dataset

predictions = model.predict(x\_test)

# Plot some example predictions to visualize the model's performance

plot\_predictions(x\_test, y\_test, predictions)