| **Practical 2**  To study and implement a Convolutional Neural Network (CNN) for image classification on the CIFAR-10 dataset. |
| --- |

| **Problem Description:** The objective is to design and implement a Convolutional Neural Network (CNN) for image classification using the CIFAR-10 dataset. After loading and normalizing the dataset, data augmentation techniques like rotations, shifts, and flips are applied to enhance diversity, which helps the model generalize better. During training, the model’s accuracy is monitored and expected to improve with each epoch. By the end, the model should demonstrate robust accuracy when tested on unseen images, showing its effectiveness in real-world image classification tasks. |
| --- |

| **Solution Architecture:** |
| --- |

| **Code:**  import tensorflow as tf  from tensorflow.keras import datasets, layers, models  from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib.pyplot as plt  from PIL import Image  from tensorflow.keras.layers import Input  from tkinter import filedialog  import numpy as np  from google.colab import files  (train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()  # Normalize pixel values to be between 0 and 1  train\_images, test\_images = train\_images / 255.0, test\_images / 255.0  class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']  data\_augmentation = ImageDataGenerator(  rotation\_range=20,  width\_shift\_range=0.2,  height\_shift\_range=0.2,  shear\_range=0.2,  zoom\_range=0.2,  horizontal\_flip=True,  fill\_mode='nearest'  )  model = models.Sequential()  model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(128, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Dropout(0.5))  model.add(layers.Flatten())  model.add(layers.Dense(64, activation='relu'))  model.add(layers.Dense(10))  model.summary()  model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True), metrics=['accuracy'])  history = model.fit(train\_images, train\_labels, epochs=10, validation\_data=(test\_images, test\_labels))  plt.plot(history.history['accuracy'], label='accuracy')  plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.ylim([0.5, 1])  plt.legend(loc='lower right')  test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)  print(test\_acc)  def preprocess\_image(image\_path):  image = Image.open(image\_path).resize((32, 32))  image = image.convert('RGB')  image = np.array(image) / 255.0 # Normalize the image  image = np.expand\_dims(image, axis=0) # Add batch dimension  return image  def predict\_image\_label(image\_path):  image = preprocess\_image(image\_path)  predictions = model.predict(image)  predicted\_label = class\_names[np.argmax(predictions)]  confidence = np.max(predictions) # Get the confidence of the prediction  return predicted\_label, confidence  print("Upload an image to classify:")  uploaded = files.upload()  # Get the number of uploaded images  num\_images = len(uploaded)  # Calculate grid dimensions (e.g., 2 rows if 2-4 images, 3 rows if 5-9 images, etc.)  num\_rows = int(np.ceil(np.sqrt(num\_images)))  num\_cols = int(np.ceil(num\_images / num\_rows))  # Create a figure and subplots  fig, axes = plt.subplots(num\_rows, num\_cols, figsize=(12, 12))  fig.subplots\_adjust(hspace=0.5) # Adjust spacing between subplots  # Flatten the axes array for easier indexing  axes = axes.flatten()  # Iterate through uploaded images and display them  for i, file\_name in enumerate(uploaded.keys()):  predicted\_label, confidence = predict\_image\_label(file\_name)  print(f"Predicted Label for {file\_name}: {predicted\_label} (Confidence: {confidence:.2f})")  # Display image on the corresponding subplot  img = Image.open(file\_name)  axes[i].imshow(img)  axes[i].set\_title(f"Predicted: {predicted\_label} ({confidence:.2f})")  axes[i].axis('off')  # Hide any unused subplots  for j in range(num\_images, num\_rows \* num\_cols):  axes[j].axis('off') |
| --- |

| **Results:**  **→ Plotted Accuracy**    **→ Accuracy** |
| --- |