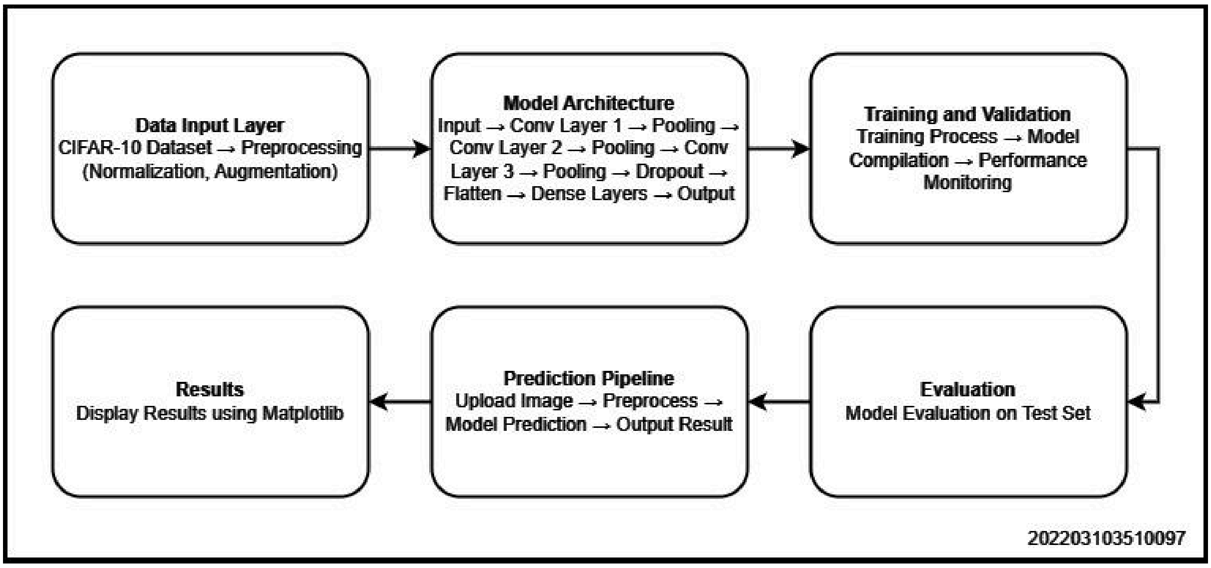
Practical 2

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:


Data Input Layer
CIFAR-10 Dataset → Preprocessing (Normalization, Augmentation)
Model Architecture
Input → Conv Layer 1 → Pooling → Conv Layer 2 → Pooling → Conv Layer 3 → Pooling → Dropout → Flatten → Dense Layers → Output
Training and Validation
Training Process → Model Compilation → Performance Monitoring
Evaluation
Model Evaluation on Test Set
Prediction Pipeline
Upload Image → Preprocess → Model Prediction → Output Result
Results
Display Results using Matplotlib
202203103510097

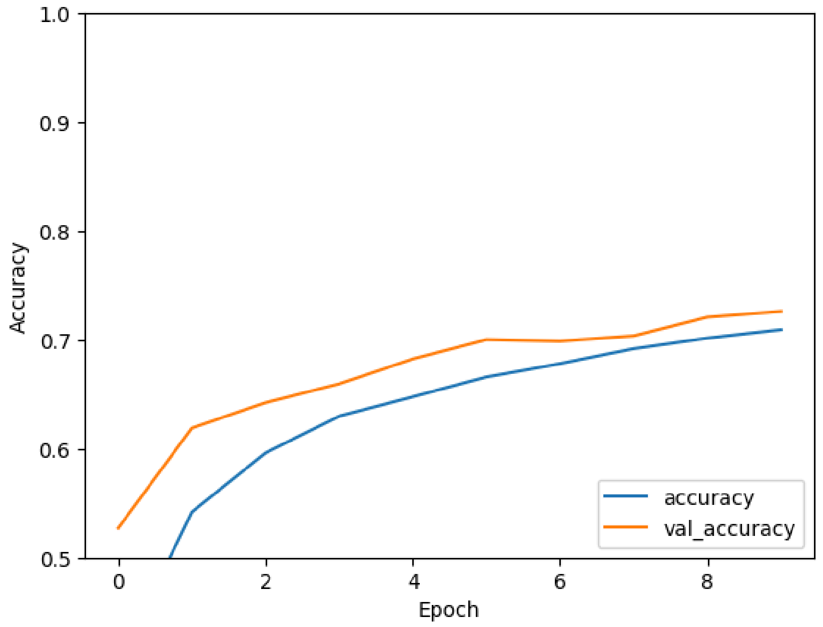
Code:

```
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')
```

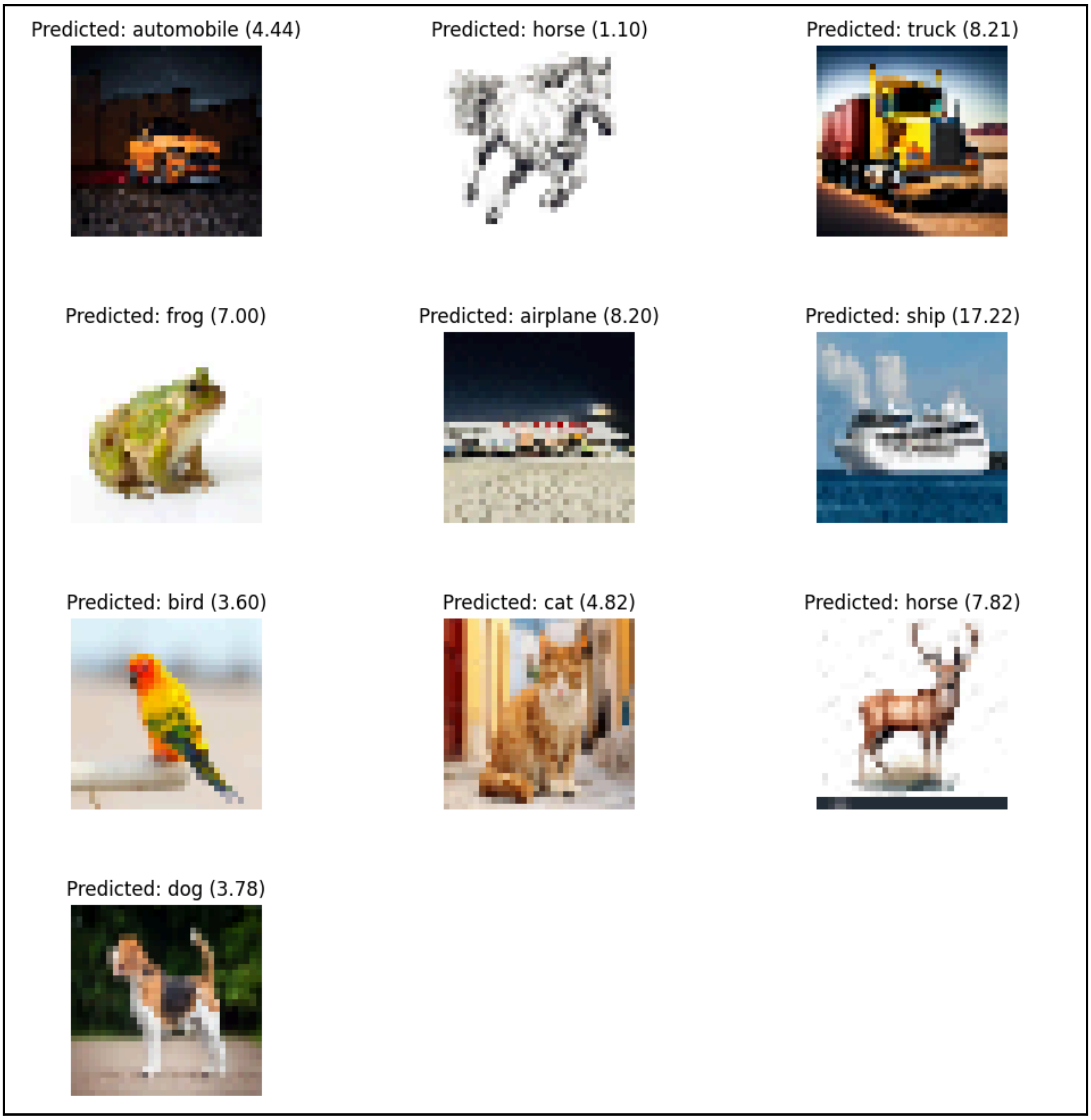
Results:

→ **Plotted Accuracy**



→ **Accuracy**

```
0.7261000275611877
```



Predicted: automobile (4.44)

Predicted: horse (1.10)

Predicted: truck (8.21)

Predicted: frog (7.00)

Predicted: airplane (8.20)

Predicted: ship (17.22)

Predicted: bird (3.60)

Predicted: cat (4.82)

Predicted: horse (7.82)

Predicted: dog (3.78)