**Image Classification with Convolutional Neural Networks (CNNs)**

**Project Overview**

**Objective**:  
The goal of this project is to develop a web application that can classify images into one of the ten classes defined by the CIFAR-10 dataset using a Convolutional Neural Network (CNN). The project encompasses data preprocessing, model training, and deployment of the model through a web interface.

**Tools and Technologies**:

* **Programming Language**: Python
* **Libraries**: TensorFlow, Keras, Flask, PIL, NumPy, Scikit-learn
* **Web Technologies**: HTML, CSS, JavaScript
* **Frontend Framework**: Flask (Python)

**1.Dataset**

**Dataset Used**:  
CIFAR-10

**Description**:  
The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 different classes, with 6,000 images per class. The dataset is divided into 50,000 training images and 10,000 testing images.

**Classes**:

* Airplane
* Automobile
* Bird
* Cat
* Deer
* Dog
* Frog
* Horse
* Ship
* Truck

**Source**:  
The dataset is publicly available and can be loaded directly using the tensorflow.keras.datasets module.

**2. Data Preprocessing**

**Image Resizing**:  
All images are resized to 32x32 pixels, which is the standard size for CIFAR-10 images.

**Normalization**:  
Pixel values of images are normalized to the range [0, 1] by dividing by 255. This helps in faster convergence during training.

**One-Hot Encoding**:  
The class labels are converted to binary class matrices using one-hot encoding, where each class label is represented as a vector with a single high bit (1) and the rest as low bits (0).

**Data Augmentation**:  
To improve model generalization, data augmentation techniques such as rotation, shifting, and flipping are applied using the ImageDataGenerator class from TensorFlow.

**3. Model Architecture**

**Model Type**:  
Convolutional Neural Network (CNN)

**Architecture Details**:

* **Input Layer**: 32x32x3 (for CIFAR-10 images)
* **Conv2D Layers**: Two convolutional layers with 32 and 64 filters, respectively. The ReLU activation function is used.
* **MaxPooling2D Layers**: Max pooling is applied after each convolutional layer to reduce the spatial dimensions.
* **Flatten Layer**: Converts the 2D matrix data to a 1D vector before feeding it to the Dense layers.
* **Dense Layers**: A fully connected layer with 128 neurons, followed by a Dropout layer for regularization.
* **Output Layer**: 10 neurons (one for each class) with a softmax activation function to output probabilities for each class.

**Regularization Techniques**:  
L2 regularization is applied to reduce overfitting, and dropout is used to further improve model robustness.

**Optimizer Used**:  
Adam optimizer, known for its adaptive learning rate, is used for model training.

**Loss Function**:  
Categorical Crossentropy is used, which is standard for multi-class classification problems.

**Learning Rate Scheduling**:  
A learning rate scheduler (ReduceLROnPlateau) is used to reduce the learning rate when the validation loss plateaus, helping in finer training adjustments.

**4. Training Process**

**Training-Testing Split**:  
The CIFAR-10 dataset comes pre-split, with 50,000 images for training and 10,000 for testing.

**Epochs**:  
The model is trained for 10 epochs.

**Batch Size**:  
64

**Training Logs**:  
Accuracy and loss are monitored over the epochs, with the model’s performance visualized through the history object.

**Augmented Training**:  
The model is trained using the augmented data generated by ImageDataGenerator.

**5. Evaluation Results**

**Accuracy**:  
The final accuracy of the model on the test data is evaluated, showing promising results.

**Classification Report**:  
A classification report generated using Scikit-learn’s classification\_report function provides precision, recall, and F1-score for each class.

**Confusion Matrix**:  
While not shown in the provided code, a confusion matrix could be generated to visualize misclassifications across classes.

**6. Deployment**

**Framework**:  
Flask

**Endpoints**:

* **/**: The home page for image uploads.
* **/predict**: Endpoint that handles image classification and returns the predicted class.

**Frontend**:  
The web interface includes an HTML form for uploading images, with a preview of the uploaded image displayed before submission. After submitting the image, the prediction is displayed on a new page.

**Backend**:  
The backend logic involves loading the trained CNN model (cnn\_cifar10\_model.keras), preprocessing the uploaded image to match the input requirements of the model, and making a prediction using the model.

**7. Deployment Instructions**

**Step 1**: Download the code

**Step 2**: Navigate to the project directory

**Step 3**: Install dependencies

**Step 4**: Run the Flask app

**Step 5**: Open the app in a web browser: Navigate to http://127.0.0.1:5000/ in your browser.

**Step 6**: Upload an image and view the prediction result.

**8. Documentation and Presentation**

**Documentation:**

* **Code Documentation**: The code includes comments explaining each major step and function. Additional docstrings could be added for more detailed explanations.
* **Project Structure**:
  + app.py: The main Flask application file.
  + templates/: Directory containing HTML templates (index.html, predict.html).
  + static/: Directory for static files like CSS, JavaScript, and images.
  + cnn\_cifar10\_model.keras: The trained model file.
  + requirements.txt: List of dependencies.

Presentation is attached in the zip file along with project code file.

**9. Conclusion**

This project demonstrates the practical application of Convolutional Neural Networks (CNNs) in image classification. It covers the entire pipeline, from data preprocessing and model training to deployment in a user-friendly web application. The use of modern web technologies and a robust deep learning framework like TensorFlow/Keras ensures that the solution is both effective and scalable.