### **Brief Report: CIFAR-10 Image Classification with Convolutional Neural Networks (CNN)**

#### **1. Data Preprocessing and Feature Engineering**

##### **Steps Taken:**

1. **Loading the CIFAR-10 Dataset**:  
   * We load the CIFAR-10 dataset using TensorFlow’s datasets.cifar10.load\_data(), which contains 60,000 32x32 color images in 10 classes, with 6,000 images per class.
2. **Normalization**:  
   * The pixel values of the images range from 0 to 255. We normalize them by dividing by 255.0, ensuring all pixel values are in the range [0, 1]. This helps the neural network train more efficiently.

python  
train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

1. **Label Encoding**:  
   * The labels are integers ranging from 0 to 9, which correspond to the 10 categories of CIFAR-10. We use SparseCategoricalCrossentropy loss, which works directly with integer labels.
2. **Feature Engineering**:  
   * Since this is an image classification task, feature engineering is minimal. The CNN architecture will automatically learn hierarchical features from the raw pixel data.

#### **2. Model Selection and Optimization Approach**

##### **Steps Taken:**

1. **Model Selection**:  
   * A **Convolutional Neural Network (CNN)** is selected due to its effectiveness in image classification tasks. The model architecture is as follows:
     + **Conv2D Layers**: Three convolutional layers, each with ReLU activation, followed by max-pooling layers. This helps in feature extraction.
     + **Flattening Layer**: Converts the 3D output of the convolutional layers into a 1D vector.
     + **Dense Layers**: A fully connected layer with 64 neurons to process the extracted features and a final output layer with 10 neurons (one for each class).

python  
model = models.Sequential([

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

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)

])

1. **Optimization**:  
   * **Optimizer**: The Adam optimizer is used, which adapts the learning rate for each parameter and is known for its efficient performance in deep learning tasks.
   * **Loss Function**: We use SparseCategoricalCrossentropy as the loss function because the task is a multi-class classification problem with integer labels.
   * **Metrics**: We use accuracy to evaluate the model performance.
   * **Training**: The model is trained for 30 epochs, with both training and validation data passed to track performance during training.

python  
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

history = model.fit(train\_images, train\_labels, epochs=30, validation\_data=(test\_images, test\_labels))

1. **Evaluation**:  
   * After training, the model's performance is evaluated on the test dataset using the model.evaluate() function.

python  
  
test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

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

#### **3. Deployment Strategy and API Usage Guide**

##### **Steps Taken:**

1. **Model Serialization (Saving the Model)**:  
   * After training, the model is saved in TensorFlow's SavedModel format using model.save(). This allows the model to be easily loaded for inference or deployment.

python  
model.save('cifar10\_model\_v1.keras')

1. **Loading the Model**:  
   * To test model loading and prediction, the saved model is reloaded using tf.keras.models.load\_model(). This ensures that the model can be successfully loaded in a deployment environment.

python  
loaded\_model = tf.keras.models.load\_model('cifar10\_model\_v1.keras')

1. **Making Predictions**:  
   * Once the model is loaded, we can make predictions on the test images. The model.predict() function is used for inference, and the predicted class for the first image is displayed.

python  
predictions = loaded\_model.predict(test\_images)

plt.imshow(test\_images[0])

plt.title(f'Predicted: {predictions[0].argmax()}')

plt.show()

1. **Visualization of Training History**:  
   * The training and validation accuracy and loss are visualized using Matplotlib to ensure the model has learned effectively and is not overfitting.

python  
plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label='val\_accuracy')

#### **API Development for Deployment**

1. **Building a REST API with Flask**:  
   * After saving and loading the model, we can serve the model predictions using a Flask API. Here’s how to set up a basic API:

python  
from flask import Flask, request, jsonify

import tensorflow as tf

import numpy as np

# Load the pre-trained model

model = tf.keras.models.load\_model('cifar10\_model\_v1.keras')

app = Flask(\_\_name\_\_)

@app.route('/predict', methods=['POST'])

def predict():

data = request.get\_json() # Get the image data from the request

image = np.array(data['image']) # Convert image to numpy array

image = np.expand\_dims(image, axis=0) # Add batch dimension

prediction = model.predict(image)

predicted\_class = np.argmax(prediction, axis=1)[0]

return jsonify({'predicted\_class': int(predicted\_class)})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

1. **Containerization with Docker**:  
   * To make the model easily deployable across different platforms, we can use Docker to containerize the Flask API. Here’s a basic Dockerfile:

dockerfile  
# Use Python 3.8 slim as the base image

FROM python:3.8-slim

# Set working directory

WORKDIR /app

# Copy requirements and install dependencies

COPY requirements.txt .

RUN pip install -r requirements.txt

# Copy the application files

COPY . .

# Expose the port on which the app will run

EXPOSE 5000

# Run the Flask application

CMD ["python", "app.py"]

A requirements.txt file containing the necessary dependencies:  
tensorflow==2.x

flask==2.x

numpy==1.x

1. **Deploying to Cloud**:  
   * Once the Docker container is built, the application can be deployed to cloud platforms such as AWS EC2, Heroku, or Google Cloud using Docker containers.

#### **GitHub Repository**

The following files will be included in the GitHub repository:

1. **train\_model.py**: Code for training the CNN model on the CIFAR-10 dataset.
2. **api.py**: Flask application for serving the trained model via a REST API.
3. **Dockerfile**: For containerizing the Flask API.
4. **requirements.txt**: Dependencies for the project.
5. **README.md**: Instructions for setting up the environment, training the model, and deploying the AP