ASSIGNMENT

TOPIC: CIFAR-10 Image Classification with CNN

BY

GINKALA DHANUSH

22STUCHH010857

SECTION-A

4th year 1st sem

CIFAR-10 Image Classification with CNN

1. Problem Statement Document

Problem Statement

Develop a Convolutional Neural Network (CNN) to classify 32×32 color images into 10 mutually exclusive categories. The goal is to build a model that learns hierarchical visual features through convolutional layers, pooling operations, and fully connected layers, thereby achieving meaningful classification accuracy on the CIFAR-10 dataset. This project demonstrates fundamental deep learning concepts, including feature extraction, pattern recognition, and model optimization.

Dataset Details

CIFAR-10 Dataset Specifications:

- Total Images: 60,000 color images
- Training Set: 50,000 images (5,000 per class)
- Test Set: 10,000 images (1,000 per class)
- Image Dimensions: 32×32 pixels with 3 RGB channels
- File Size: 163 MB (compressed Python version)
- Classes: 10 mutually exclusive categories

Class Distribution:

Index	Class Name	Training Images	Test Images	Description
0	Airplane	5,000	1,000	Various aircraft types
1	Automobile	5,000	1,000	Cars, sedans, SUVs (excluding trucks)
2	Bird	5,000	1,000	Different bird species
3	Cat	5,000	1,000	Various cat breeds
4	Deer	5,000	1,000	Deer and similar animals
5	Dog	5,000	1,000	Various dog breeds
6	Frog	5,000	1,000	Frogs and amphibians
7	Horse	5,000	1,000	Horses and ponies
8	Ship	5,000	1,000	Ships and boats
9	Truck	5,000	1,000	Large trucks (excluding pickups)

Dataset Links:

- Official Website:
- https://www.cs.toronto.edu/~kriz/cifar.html
- TensorFlow Datasets:
- https://www.tensorflow.org/datasets/catalog/cifar10
- Direct Download:
- https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
- Kaggle Version:
- https://www.kaggle.com/datasets/ahmedhessam/cifar10

Dataset Characteristics:

- Images are low-resolution (32×32), making them computationally efficient
- Contains significant intra-class variation and background clutter
- Represents real-world objects with various orientations and lighting conditions
- Widely used benchmark for computer vision research

Python Modules (Dependencies)

Module	Version	Purpose
tensorflow	≥2.8.0	Deep learning framework, model building, training
numpy	≥1.21.0	Numerical operations, array handling
matplotlib	≥3.5.0	Visualization, plotting training curves and images

Python Code File

```
"""
CIFAR-10 Image Classification with Convolutional Neural Network
Author: Deep Learning Project
Date: November 2025
"""

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import time
import os

def load_and_preprocess_data():
    """
    Load CIFAR-10 dataset and normalize pixel values
    Returns: (x_train, y_train), (x_test, y_test)
    """
    print("="*60)
    print("STEP 1: Loading CIFAR-10 Dataset")
    print("="*60)

    # Load dataset from Keras (automatically downloads if not present)
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
```

```
    print(f"Training data shape: {x_train.shape} - {len(x_train)} images")
    print(f"Test data shape: {x_test.shape} - {len(x_test)} images")

    # Normalize pixel values from [0, 255] to [0, 1]
    print("\nNormalizing pixel values...")
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    return (x_train, y_train), (x_test, y_test)

def create_cnn_model():
    """
    Build Convolutional Neural Network architecture
    Returns: Compiled CNN model
    """
    print("\n" + "="*60)
    print("STEP 2: Building CNN Architecture")
    print("="*60)

    model = tf.keras.Sequential([
        # First Convolutional Block
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                               input_shape=(32, 32, 3),
                               name='conv2d_1'),
        tf.keras.layers.MaxPooling2D((2, 2), name='max_pool_1'),
```

```
        # Second Convolutional Block
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu', name='conv2d_2'),
        tf.keras.layers.MaxPooling2D((2, 2), name='max_pool_2'),

        # Third Convolutional Block
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu', name='conv2d_3'),

        # Fully Connected Layers
        tf.keras.layers.Flatten(name='flatten'),
        tf.keras.layers.Dense(64, activation='relu', name='dense_1'),
        tf.keras.layers.Dense(10, activation='softmax', name='output')
    ])

    # Compile model
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    print("Model Architecture:")
    model.summary()

    return model

def train_model(model, x_train, y_train, x_test, y_test):
```

```
"""
Train the CNN model with training data
"""
print("\n" + "="*60)
print("STEP 3: Training the Model")
print("="*60)

# Training parameters
epochs = 10
batch_size = 64

print(f"Training Parameters:")
print(f"  Epochs: {epochs}")
print(f"  Batch Size: {batch_size}")
print(f"  Training Samples: {len(x_train)}")
print(f"  Validation Samples: {len(x_test)}")

# Train model with timing
start_time = time.time()
history = model.fit(
    x_train, y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(x_test, y_test),
    verbose=1
)
```

```
    training_time = time.time() - start_time

    print(f"\nTraining completed in {training_time:.2f} seconds")

    return history, training_time

def evaluate_model(model, x_test, y_test):
    """
    Evaluate trained model on test set
    """
    print("\n" + "="*60)
    print("STEP 4: Evaluating Model Performance")
    print("="*60)

    # Evaluate on test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)

    print(f"Test Loss: {test_loss:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")
    print(f"Test Accuracy (percentage): {test_accuracy*100:.2f}%")

    return test_loss, test_accuracy

def visualize_predictions(model, x_test, y_test, class_names):
    """
    Visualize sample predictions with actual vs predicted labels
```

```
    """
    print("\n" + "="*60)
    print("STEP 5: Generating Sample Predictions")
    print("="*60)

    # Make predictions on first 10 test images
    predictions = model.predict(x_test[:10], verbose=0)
    predicted_classes = np.argmax(predictions, axis=1)

    print("\nSample Predictions (first 10 images):")
    print("-" * 50)

    for i in range(10):
        true_label = class_names[y_test[i][0]]
        pred_label = class_names[predicted_classes[i]]
        correct = "✓" if y_test[i][0] == predicted_classes[i] else "✗"
            print(f"Image  {i+1:2d}:  True={true_label:10s} | Predicted={pred_label:10s} |
{correct}")

    return predictions

def plot_training_history(history):
    """
    Plot training and validation accuracy/loss curves
    """
    print("\n" + "="*60)
```

```
print("STEP 6: Plotting Training History")
print("="*60)

# Create output directory
os.makedirs('project_outputs', exist_ok=True)

# Plot accuracy
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

```
    plt.grid(True)

    plt.tight_layout()
    plt.savefig('project_outputs/training_history.png', dpi=300, bbox_inches='tight')
    print("Training history plot saved to: project_outputs/training_history.png")
    plt.close()

def save_trained_model(model):
    """
    Save the trained model to disk
    """
    print("\n" + "="*60)
    print("STEP 7: Saving Model")
    print("="*60)

    # Save model in H5 format
    model.save('project_outputs/cifar10_cnn_model.h5')
    print("Model saved to: project_outputs/cifar10_cnn_model.h5")

    # Save model architecture as JSON
    model_json = model.to_json()
    with open('project_outputs/model_architecture.json', 'w') as f:
        f.write(model_json)
    print("Model architecture saved to: project_outputs/model_architecture.json")

def display_sample_images(x_test, y_test, class_names, num_images=5):
```

```
    """
    Display sample test images with their true labels
    """
    print("\n" + "="*60)
    print("STEP 8: Displaying Sample Images")
    print("="*60)

    plt.figure(figsize=(10, 2))
    for i in range(num_images):
        plt.subplot(1, num_images, i+1)
        plt.imshow(x_test[i])
        plt.title(f"True: {class_names[y_test[i][0]]}")
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('project_outputs/sample_images.png', dpi=300, bbox_inches='tight')
    print(f"Sample images saved to: project_outputs/sample_images.png")
    plt.close()

def main():
    """
    Main execution function
    """
    print("\n" + "="*60)
    print("CIFAR-10 CNN CLASSIFICATION PROJECT")
    print("="*60)
```

```
# Define class names
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Step 1: Load data
(x_train, y_train), (x_test, y_test) = load_and_preprocess_data()

# Step 2: Create model
model = create_cnn_model()

# Step 3: Train model
history, training_time = train_model(model, x_train, y_train, x_test, y_test)

# Step 4: Evaluate model
test_loss, test_accuracy = evaluate_model(model, x_test, y_test)

# Step 5: Visualize predictions
visualize_predictions(model, x_test, y_test, class_names)

# Step 6: Plot training history
plot_training_history(history)

# Step 7: Save model
save_trained_model(model)

# Step 8: Display sample images
```

```
    display_sample_images(x_test, y_test, class_names)


    # Print final summary
    print("\n" + "="*60)
    print("PROJECT EXECUTION SUMMARY")
    print("="*60)
    print(f"Test Accuracy: {test_accuracy*100:.2f}%")
    print(f"Test Loss: {test_loss:.4f}")
        print(f"Total Training Time: {training_time:.2f} seconds ({training_time/60:.2f} minutes)")
    print(f"All outputs saved in: project_outputs/")
    print("="*60)


if __name__ == "__main__":
    main()
```