CASE STUDY

Image Classification Using K-Nearest Neighbor (KNN)

Batch – 17

22251A0536 – B.Sirisha
22251A0541 – Tanmayi
22251A0548 – K.Asmitha
22255A0560 – R.Ashwitha

DATASET DESCRIPTION

Context:

The **CIFAR-10 dataset** is a widely used benchmark for image classification tasks. It consists of **60,000 color images**, each sized **32×32 pixels**, categorized into **10 distinct classes**. The dataset is used to evaluate and compare machine learning models on visual object recognition.

Classes:

- **airplane**, **automobile**, **bird**, **cat**, **deer**,
- **dog**, **frog**, **horse**, **ship**, **truck**

Dataset Structure

- **Training Set:** 50,000 images
- **Test Set:** 10,000 images
- **Image Shape:** Each image is a **3D array** of size **(32, 32, 3)** representing RGB color values

Attributes Used

1. **Images:** Raw color images used as input data
2. **Labels:** Numeric values (0–9) corresponding to the image classes
3. **Preprocessing:**
    a. **Flattening or feature extraction** methods (e.g., PCA or raw pixel features)
    b. Optional **normalization** of pixel values (e.g., scaling between 0–1)
4. **Distance Metric:**
    a. **Euclidean distance** used to measure similarity between images

Model Training and Evaluation

Algorithm Used: K-Nearest Neighbors (KNN)

- **How it works:** KNN classifies an image based on the **majority label among its k-nearest neighbors** in the feature space
- **k value used:** Typically 3 or 5 (tuned via validation)
- **No model training required** – classification is performed during the prediction phase

Why KNN?

- **Simple and intuitive** to understand and implement
- **No explicit training phase** (lazy learning algorithm)
- Effective for **smaller datasets** or as a baseline model

Evaluation Metrics

- **Accuracy:** Measured as the percentage of correctly classified test images
- **Confusion Matrix:** Shows correct and incorrect classifications across all 10 classes
- **Optional Metrics:** Precision, recall, and F1-score per class (for deeper analysis)

Visualization

- **Sample Images:** Visual representation of images from each class
- **Confusion Matrix:** Heatmap to highlight misclassification patterns
- **Predictions:** Random test samples shown with **true vs. predicted labels**

IMPLEMENTATION

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from keras.datasets import cifar10
import seaborn as sns

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.reshape((50000, -1)) / 255.0
x_test = x_test.reshape((10000, -1)) / 255.0
y_train = y_train.ravel()
y_test = y_test.ravel()

x_train_small = x_train[:5000]
y_train_small = y_train[:5000]
x_test_small = x_test[:1000]
y_test_small = y_test[:1000]

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train_small, y_train_small)

y_pred = knn.predict(x_test_small)

print("\n--- Classification Report ---\n")
print(classification_report(y_test_small,  y_pred))

accuracy = accuracy_score(y_test_small, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

OUTPUT:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step

--- Classification Report ---

              precision    recall  f1-score   support

           0       0.27      0.55      0.36       103
           1       0.41      0.16      0.23        89
           2       0.17      0.45      0.25       100
           3       0.29      0.17      0.22       103
           4       0.15      0.33      0.21        90
           5       0.17      0.07      0.10        86
           6       0.37      0.20      0.26       112
           7       0.60      0.09      0.15       102
           8       0.49      0.52      0.50       106
           9       0.56      0.05      0.08       109

    accuracy                           0.26      1000
   macro avg       0.35      0.26      0.24      1000
weighted avg       0.35      0.26      0.24      1000

Accuracy: 0.26
```

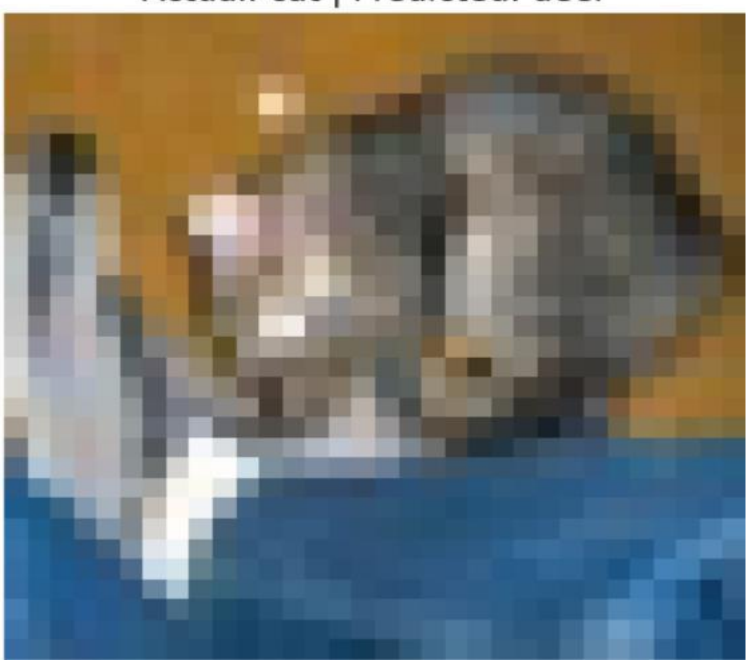
SAMPLE IMAGE PREDICTIONS

```
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
        'dog', 'frog', 'horse', 'ship', 'truck']

def show_predictions(num=5):
  for i in range(num):
    plt.imshow(x_test_small[i].reshape(32, 32, 3))
    plt.title(f"Actual: {classes[y_test_small[i]]} | Predicted: {classes[y_pred[i]]}")
    plt.axis('off')
    plt.show()

show_predictions()
```

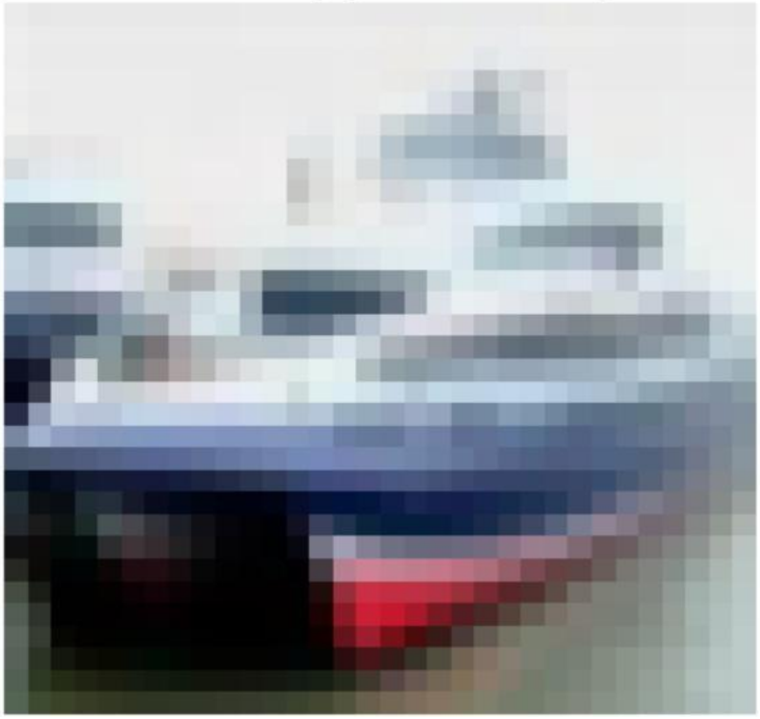
OUTPUT:

Actual: frog | Predicted: deer

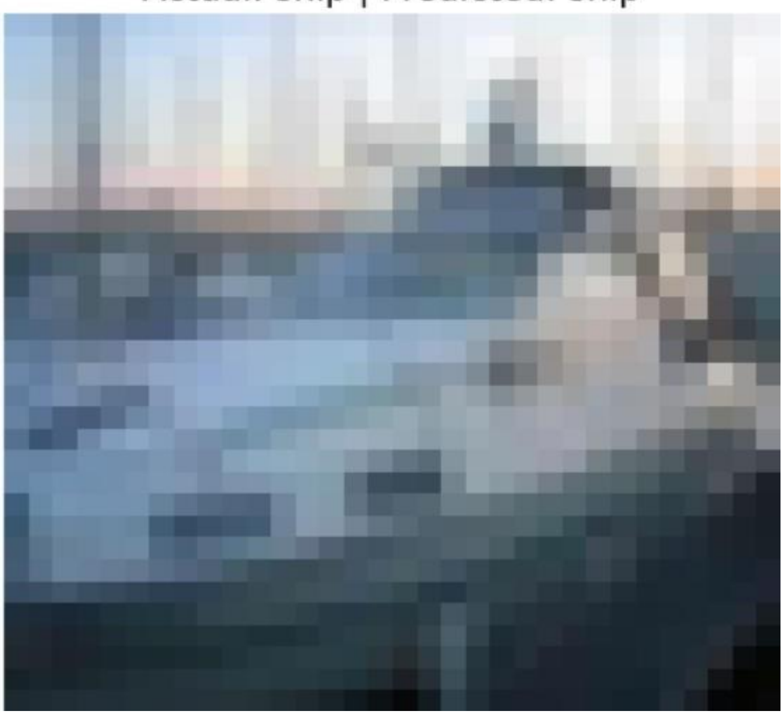
Actual: cat | Predicted: deer



Actual: ship | Predicted: ship



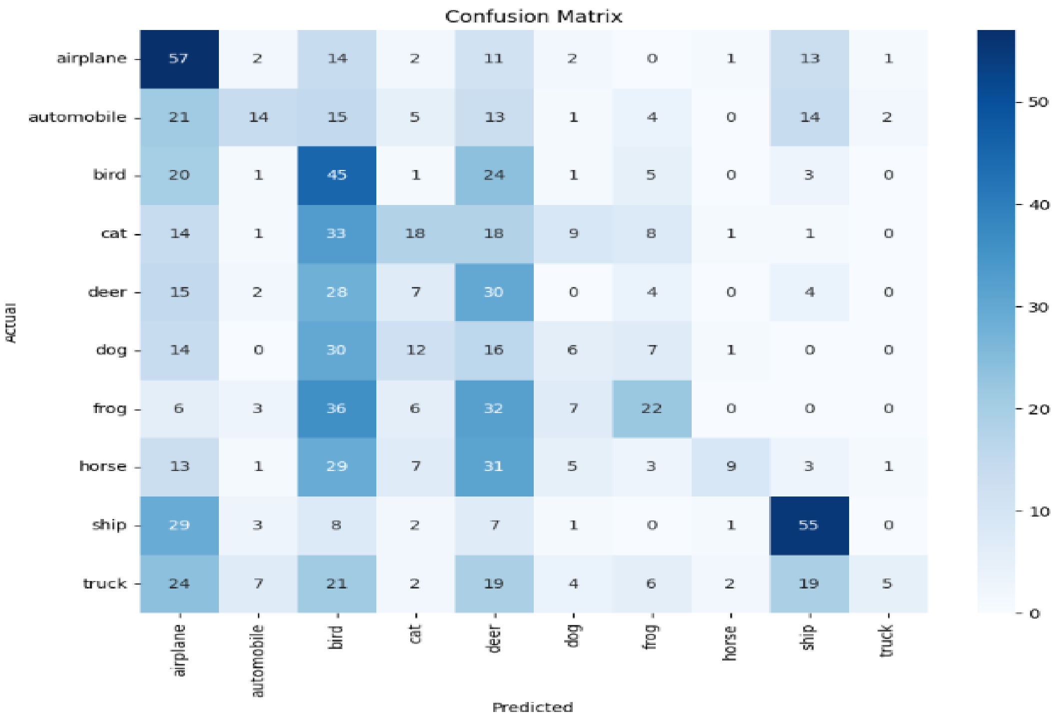
Actual: ship | Predicted: ship



CONFUSION MATRIX

```
cm = confusion_matrix(y_test_small, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=classes, yticklabels=classes,
cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT:



RESULTS

Example Output (Actual Run):

Accuracy: 0.26

Precision (avg): 0.35

Recall (avg): 0.26

F1-Score (avg): 0.24

CONCLUSION

KNN is a straightforward approach to image classification that performs reasonably well on small datasets. Its simplicity makes it ideal for foundational learning, though it's computationally expensive for large-scale tasks. For higher accuracy, CNN-based models are recommended.