Name: Harsh Patil
Class: D15C/37

Experiment No. 8

AIM: Design and train a Convolutional Neural Network (CNN) on image datasets (CIFAR-10)

1. Dataset Source

The dataset used for this experiment is the **CIFAR-10**, a standard benchmark for image classification.

- **Dataset Link:** Keras Datasets - CIFAR10
- **Access:** Directly loaded via tensorflow.keras.datasets.

2. Dataset Description

The CIFAR-10 dataset consists of color images used for object recognition.

- **Number of instances:** 60,000 images (50,000 training, 10,000 testing).
- **Number of features:** 32×32 pixel images with 3 color channels (RGB).
- **Target Variable:** 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).
- **Characteristics:** The images are low-resolution, which requires the model to learn spatial patterns rather than just raw pixel intensities.

3. Mathematical Formulation

CNNs differ from ANNs by utilizing **Convolution** and **Pooling** layers to preserve spatial structure.

3.1 Convolution Operation

The core of a CNN is the discrete convolution, where a kernel (filter) K slides over image I:

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

This allows the network to learn local patterns like edges and textures through shared weights.

3.2 Activation and Downsampling

- **ReLU:** Applied to introduce non-linearity: $f(x) = \max(0, x).$
- **Max Pooling:** Reduces spatial dimensions by taking the maximum value in a window (e.g 2 x 2), reducing computational load and providing translation invariance.

4. Algorithm Limitations

- **Computational Intensity:** Training CNNs requires significantly more GPU power and memory compared to standard ANNs.
- **Data Requirement:** CNNs are prone to overfitting without a massive volume of varied data or data augmentation techniques.

- **Resolution Sensitivity:** They struggle with images of varying sizes unless they are resized to a uniform input dimension.

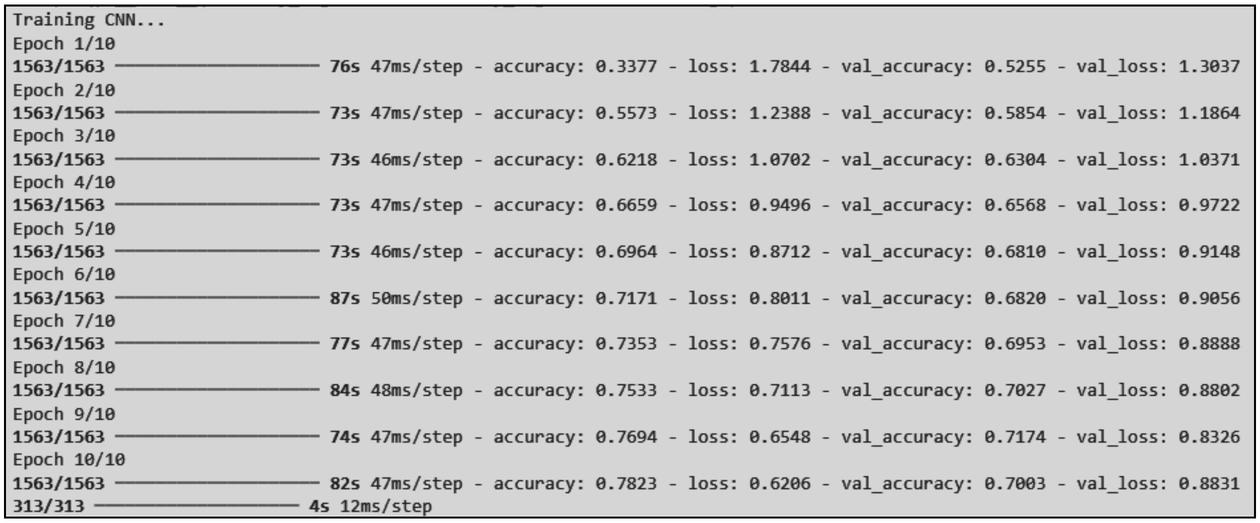
5. Methodology / Workflow

1. **Data Acquisition:** Load CIFAR-10 data via TensorFlow.
2. **Preprocessing:** Normalize pixel values to the range [0, 1] by dividing by 255 to ensure stable gradient descent.
3. **Feature Extraction:** Stack Conv2D layers for pattern recognition and MaxPooling2D layers for spatial reduction.
4. **Classification:** Flatten the 3D feature maps into a 1D vector and pass through Dense layers.
5. **Compilation:** Use **Adam** optimizer and **Sparse Categorical Cross-Entropy** for multi-class optimization.
6. **Evaluation:** Visualize training history and use a Confusion Matrix to identify class confusion.

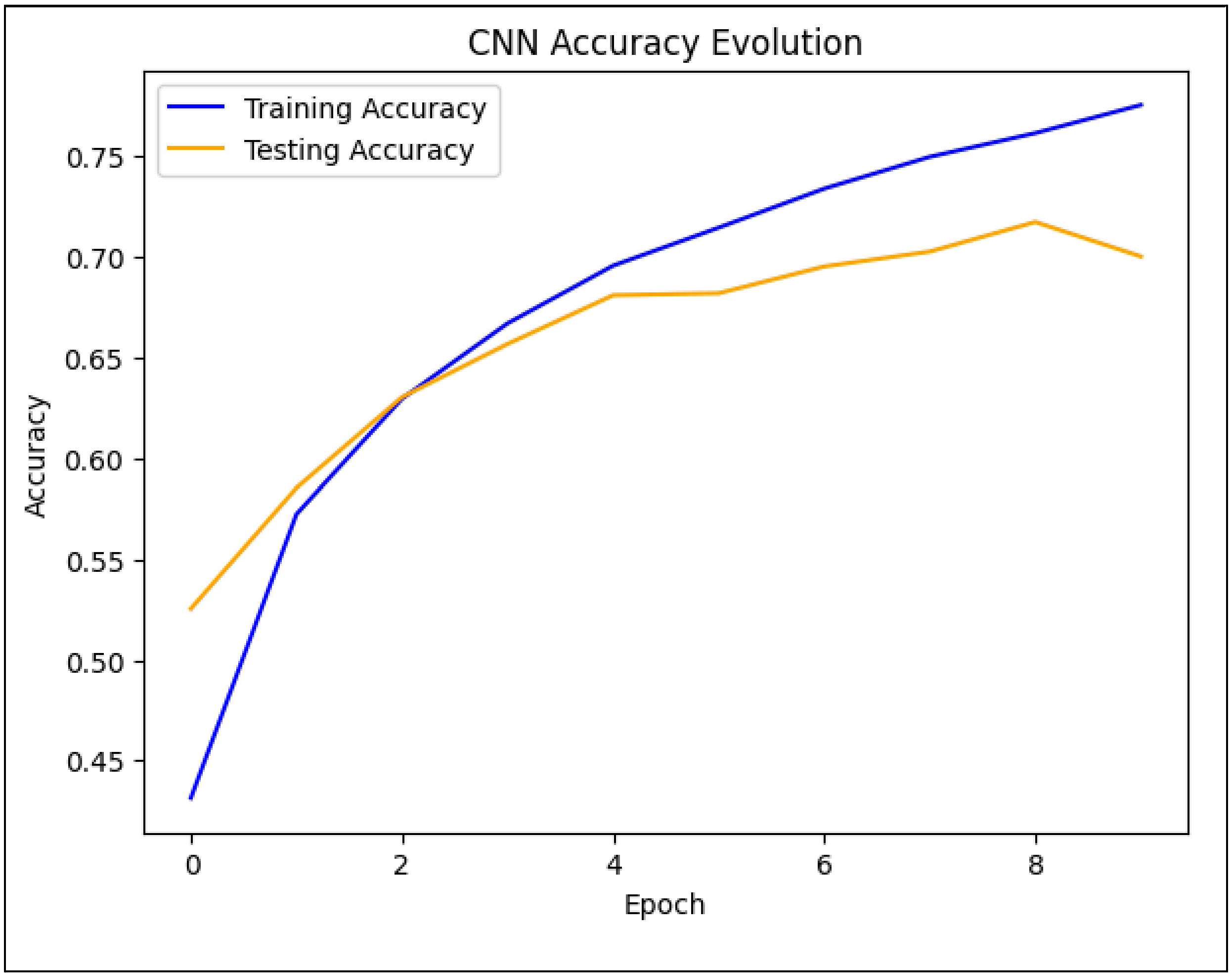
6. Implementation Code

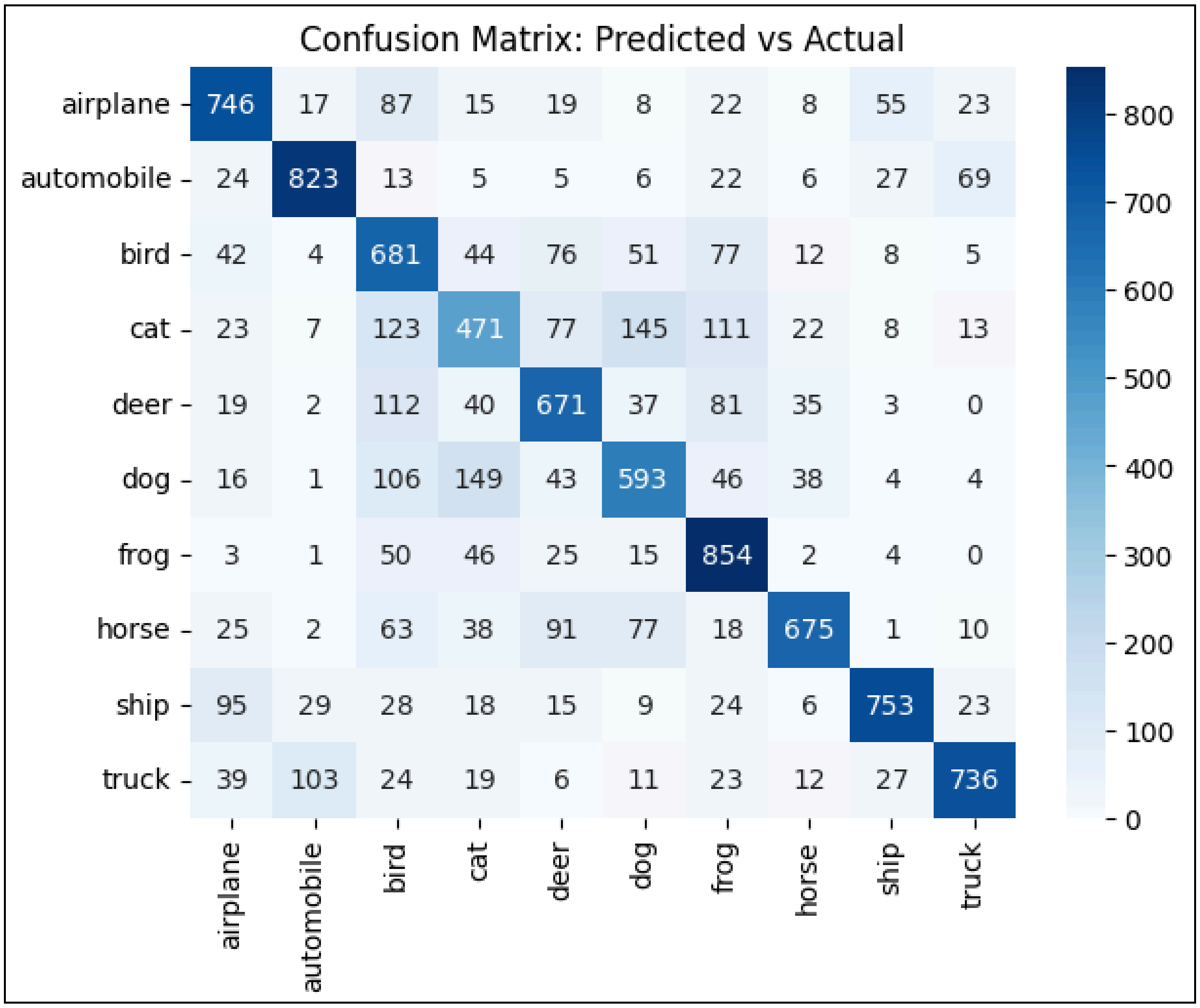
```
import tensorflow as tf

from tensorflow.keras import
datasets, layers, models

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from sklearn.metrics import
confusion_matrix


# 1. Load and Normalize Dataset

(train_images, train_labels),
(test_images, test_labels) =
datasets.cifar10.load_data()

train_images, test_images =
train_images / 255.0, test_images /
255.0

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


# 2. Build the CNN Architecture

model = models.Sequential([

    # Convolutional Base

    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'),



    # Dense Classifier
    layers.Flatten(),

    layers.Dense(64,
activation='relu'),

    layers.Dense(10) # 10 output
classes

])



# 3. Compile and Train

model.compile(optimizer='adam',


loss=tf.keras.losses.SparseCategoric
alCrossentropy(from_logits=True),

              metrics=['accuracy'])



history = model.fit(train_images,
train_labels, epochs=10,


validation_data=(test_images,
test_labels), verbose=1)



# 4. Visual Performance Analysis

plt.figure(figsize=(12, 5))



# Accuracy Plot

plt.subplot(1, 2, 1)

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

plt.plot(history.history['val_accura
cy'], label='Validation Accuracy')

plt.title('CNN Accuracy Evolution')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()



# Confusion Matrix

plt.subplot(1, 2, 2)

y_pred =
np.argmax(model.predict(test_images)
, axis=1)

cm = confusion_matrix(test_labels,
y_pred)

sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues',
xticklabels=class_names,
yticklabels=class_names)

plt.title('Confusion Matrix')

plt.show()



print("\nFinal Test Accuracy:",
model.evaluate(test_images,
test_labels, verbose=0)[1])
```

OUTPUT:

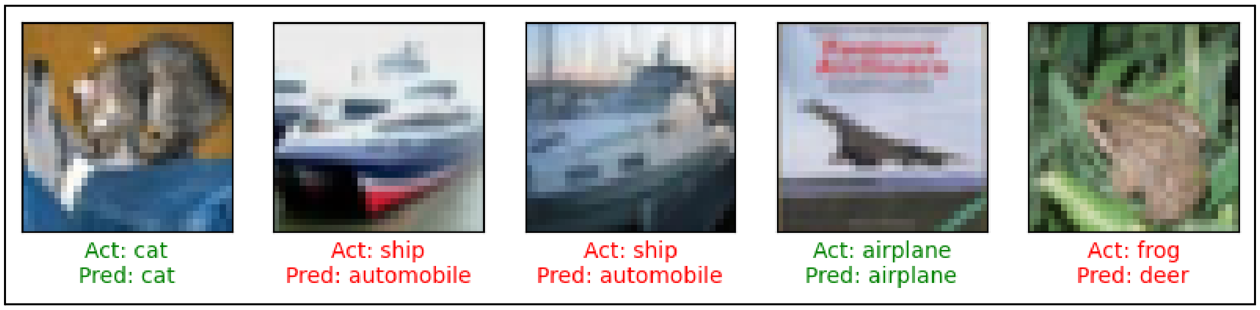


```
Training CNN...
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 47ms/step - accuracy: 0.3377 - loss: 1.7844 - val_accuracy: 0.5255 - val_loss: 1.3037
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 73s 47ms/step - accuracy: 0.5573 - loss: 1.2388 - val_accuracy: 0.5854 - val_loss: 1.1864
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 73s 46ms/step - accuracy: 0.6218 - loss: 1.0702 - val_accuracy: 0.6304 - val_loss: 1.0371
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 73s 47ms/step - accuracy: 0.6659 - loss: 0.9496 - val_accuracy: 0.6568 - val_loss: 0.9722
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 73s 46ms/step - accuracy: 0.6964 - loss: 0.8712 - val_accuracy: 0.6810 - val_loss: 0.9148
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 87s 50ms/step - accuracy: 0.7171 - loss: 0.8011 - val_accuracy: 0.6820 - val_loss: 0.9056
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 77s 47ms/step - accuracy: 0.7353 - loss: 0.7576 - val_accuracy: 0.6953 - val_loss: 0.8888
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 84s 48ms/step - accuracy: 0.7533 - loss: 0.7113 - val_accuracy: 0.7027 - val_loss: 0.8802
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 74s 47ms/step - accuracy: 0.7694 - loss: 0.6548 - val_accuracy: 0.7174 - val_loss: 0.8326
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 47ms/step - accuracy: 0.7823 - loss: 0.6206 - val_accuracy: 0.7003 - val_loss: 0.8831
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step
```





	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	746	17	87	15	19	8	22	8	55	23
automobile	24	823	13	5	5	6	22	6	27	69
bird	42	4	681	44	76	51	77	12	8	5
cat	23	7	123	471	77	145	111	22	8	13
deer	19	2	112	40	671	37	81	35	3	0
dog	16	1	106	149	43	593	46	38	4	4
frog	3	1	50	46	25	15	854	2	4	0
horse	25	2	63	38	91	77	18	675	1	10
ship	95	29	28	18	15	9	24	6	753	23
truck	39	103	24	19	6	11	23	12	27	736



7. Performance Analysis

The CNN achieved a significantly higher accuracy on image data compared to traditional fully connected networks.

- **Observations:** The **Convolutional layers** successfully acted as automated feature extractors. The **Validation Accuracy** curves show that while the model learns quickly, it begins to plateau around epoch 10, indicating that further training might require data augmentation.

- **Class Confusion:** The Confusion Matrix indicates that the model occasionally confuses semantically similar classes, such as "cats" and "dogs," which share similar pixel-level features.

8. Conclusion

This experiment validates the superiority of Convolutional Neural Networks for image-related tasks. By utilizing spatial filters and max-pooling, the model effectively captured the hierarchical nature of image data, from simple edges to complex object shapes. The shift from manual feature engineering to automated feature extraction via Keras demonstrates the power of CNNs in providing scalable and accurate solutions for computer vision challenges.