# Image Classification Using Convolutional Neural Networks (CNN)

## Introduction

Image classification is a fundamental task in computer vision that involves assigning a label to an image based on its content. Convolutional Neural Networks (CNNs) prove to be highly effective for this task due to their ability to automatically learn spatial hierarchies of features from images. I have used CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images.

## Problem Definition

Given an image, the task is to predict the correct class of this image. The images are very small (32x32), making it challenging to distinguish them even for a human. In this notebook, we build a CNN model that classifies images of various objects across 10 classes:

1. Airplane
2. Automobile
3. Bird
4. Cat
5. Deer
6. Dog
7. Frog
8. Horse
9. Ship
10. Truck

## Evaluation

With 10 classes, if we randomly guess the class of an image, we have a 1/10 probability of being correct.

## Data Loading

We load the CIFAR-10 dataset using TensorFlow.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import tensorflow as tf

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

print(f"X\_train shape: {X\_train.shape}")

print(f"y\_train shape: {y\_train.shape}")

print(f"X\_test shape: {X\_test.shape}")

print(f"y\_test shape: {y\_test.shape}")

## Data Visualization

We visualize the dataset to understand the distribution of classes.

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

W\_grid = 10

L\_grid = 10

fig, axes = plt.subplots(L\_grid, W\_grid, figsize=(17, 17))

axes = axes.ravel()

n\_train = len(X\_train)

for i in np.arange(0, W\_grid \* L\_grid):

index = np.random.randint(0, n\_train)

axes[i].imshow(X\_train[index, 1:])

label\_index = int(y\_train[index])

axes[i].set\_title(labels[label\_index], fontsize=8)

axes[i].axis('off')

plt.subplots\_adjust(hspace=0.4)

## Data Preprocessing

We scale the data and transform the target variable into one-hot encoding.

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

y\_cat\_train = to\_categorical(y\_train, 10)

y\_cat\_test = to\_categorical(y\_test, 10)

## Model Building

We build a CNN model using TensorFlow's Keras API.

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.preprocessing.image import ImageDataGenerator

early\_stop = EarlyStopping(monitor='val\_loss', patience=2)

batch\_size = 32

data\_generator = ImageDataGenerator(width\_shift\_range=0.1, height\_shift\_range=0.1, horizontal\_flip=True)

train\_generator = data\_generator.flow(X\_train, y\_cat\_train, batch\_size=batch\_size)

steps\_per\_epoch = X\_train.shape[0] // batch\_size

model = Sequential()

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

model.add(MaxPool2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

r = model.fit(train\_generator, epochs=50, steps\_per\_epoch=steps\_per\_epoch, validation\_data=(X\_test, y\_cat\_test), callbacks=[early\_stop])

## Model Evaluation

We evaluate the model's performance on the test dataset.

plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)

plt.plot(r.history['loss'], label='Loss')

plt.plot(r.history['val\_loss'], label='val\_Loss')

plt.title('Loss Function Evolution')

plt.legend()

plt.subplot(4, 2, 2)

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

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

plt.title('Accuracy Function Evolution')

plt.legend()

evaluation = model.evaluate(X\_test, y\_cat\_test)

print(f'Test Accuracy : {evaluation[1] \* 100:.2f}%')

## Testing on One Image

We test the model on a specific image from the test set.

my\_image = X\_test[100]

plt.imshow(my\_image)

print(f" Image 100 is {y\_test[100]}")

pred\_100 = np.argmax(model.predict(my\_image.reshape(1, 32, 32, 3)))

print(f"The model predicts that image 100 is {pred\_100}")

## Save the Model

Finally, we save the trained model for future use.

from tensorflow.keras.models import load\_model

model.save('cnn\_50\_epochs.h5')

This documentation provides a comprehensive overview of the image classification task using a CNN model.

**Name: Suyog Acharya**

**Email: acharyasuyog09@gmail.com**