# 2. BUILD A SIMPLE CNN MODEL FOR IMAGE SEGMENTATION

|  |  |
| --- | --- |
| **EX.N0 : 2** | **BUILD A SIMPLE CNN MODEL FOR IMAGE SEGMENTATION** |
| **DATE : 03/02/2025** |

**AIM:**

To build and train a simple Convolutional Neural Network (CNN) for performing binary image segmentation using the CIFAR-10 dataset.

# ALGORITHM:

Step 1: Import necessary libraries like TensorFlow, NumPy, and Matplotlib. Step 2: Load the CIFAR-10 dataset and normalize the images.

Step 3: Create binary segmentation masks using a threshold on image brightness. Step 4: Design a simple CNN-based encoder-decoder architecture for segmentation. Step 5: Compile the model using binary cross entropy loss and accuracy metric.

Step 6: Train the model with training images and corresponding masks. Step 7: Evaluate the model using test data.

Step 8: Visualize the original image, ground truth mask, and predicted segmentation mask.

# PROGRAM:

import tensorflow as tf

from tensorflow.keras import layers, models import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import cifar10

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

x\_train = x\_train.astype("float32") / 255.0 x\_test = x\_test.astype("float32") / 255.0

y\_train\_segmentation = np.where(x\_train.mean(axis=-1, keepdims=True) > 0.5, 1, 0)

y\_test\_segmentation = np.where(x\_test.mean(axis=-1, keepdims=True) > 0.5, 1, 0) def create\_segmentation\_model(input\_shape):

model = models.Sequential([ layers.InputLayer(input\_shape=input\_shape), layers.Conv2D(32, (3, 3), activation="relu", padding="same"),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation="relu", padding="same"),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(128, (3, 3), activation="relu", padding="same"),

layers.Conv2DTranspose(64, (3, 3), strides=2, activation="relu", padding="same"),

layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same"),

layers.Conv2D(1, (1, 1), activation="sigmoid", padding="same")

])

return model

input\_shape = x\_train.shape[1:] # (32, 32, 3) model = create\_segmentation\_model(input\_shape)

model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"]) model.summary()

history = model.fit(

x\_train, y\_train\_segmentation, validation\_data=(x\_test, y\_test\_segmentation), epochs=5,

batch\_size=32

)

loss, accuracy = model.evaluate(x\_test, y\_test\_segmentation) print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}") predictions = model.predict(x\_test)

num\_images = 3

plt.figure(figsize=(12, num\_images \* 4)) for i in range(num\_images): plt.subplot(num\_images, 3, i \* 3 + 1) plt.title("Original Image") plt.imshow(x\_test[i])

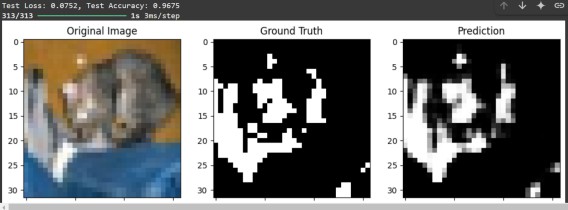
plt.axis('off') plt.subplot(num\_images, 3, i \* 3 + 2) plt.title("Ground Truth")

plt.imshow(y\_test\_segmentation[i].squeeze(), cmap="gray") plt.axis('off')

plt.subplot(num\_images, 3, i \* 3 + 3) plt.title("Prediction") plt.imshow(predictions[i].squeeze(), cmap="gray") plt.axis('off')

plt.tight\_layout() plt.show()

# OUTPUT:

****

**RESULT:**

Thus, the Program has been executed successfully and verified.