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.

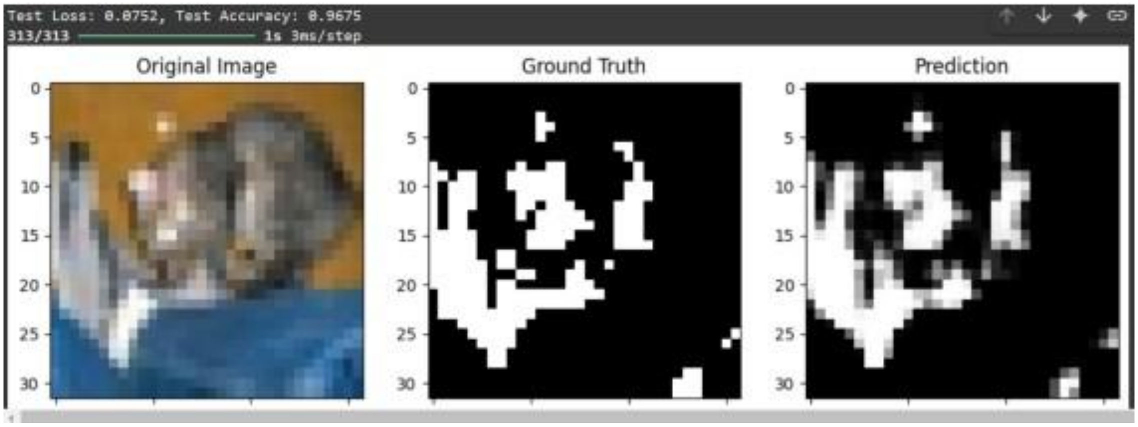
PROGRAM:

```
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:



```
Test Loss: 0.0752, Test Accuracy: 0.9675
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

RESULT:

Thus, the Program has been executed successfully and verified.