**SEMANTUC UNET**

import numpy as np

import tensorflow as tf

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.datasets import cifar10

# Load CIFAR-10 data

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

# Preprocess images

x\_train = x\_train / 255.0 # Normalize images

x\_test = x\_test / 255.0

# Create dummy segmentation masks (same shape as input)

# Replace this with real masks if available

y\_train\_seg = (x\_train.mean(axis=-1) > 0.5).astype(int) # Example threshold mask

y\_test\_seg = (x\_test.mean(axis=-1) > 0.5).astype(int)

# Convert masks to categorical for multiple classes (if applicable)

#y\_train\_seg = y\_train\_seg[..., np.newaxis] # Add channel dimension

#y\_test\_seg = y\_test\_seg[..., np.newaxis]

y\_train\_seg = y\_train\_seg[:, :, :, np.newaxis] # Add channel dimension

y\_test\_seg = y\_test\_seg[:, :, :, np.newaxis] # Add channel dimension

from tensorflow.keras import Model, Input

from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D, concatenate

def unet\_model(input\_size=(32, 32, 3)):

inputs = Input(input\_size)

# Downsampling

c1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)

p1 = MaxPooling2D((2, 2))(c1)

c2 = Conv2D(64, (3, 3), activation='relu', padding='same')(p1)

p2 = MaxPooling2D((2, 2))(c2)

# Bottleneck

c3 = Conv2D(128, (3, 3), activation='relu', padding='same')(p2)

# Upsampling

u1 = UpSampling2D((2, 2))(c3)

m1 = concatenate([u1, c2])

c4 = Conv2D(64, (3, 3), activation='relu', padding='same')(m1)

u2 = UpSampling2D((2, 2))(c4)

m2 = concatenate([u2, c1])

c5 = Conv2D(32, (3, 3), activation='relu', padding='same')(m2)

outputs = Conv2D(1, (1, 1), activation='sigmoid')(c5)

return Model(inputs, outputs)

# Compile the model

model = unet\_model()

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

# Train the model

model.fit(x\_train, y\_train\_seg, validation\_data=(x\_test, y\_test\_seg), epochs=10, batch\_size=32)

import matplotlib.pyplot as plt

pred = model.predict(x\_test[6:11]) # Select images from index 6 to 10

# Display images and masks

for i in range(6, 11): # Iterate from index 6 to 10

plt.subplot(1, 3, 1)

plt.title("Input Image")

plt.imshow(x\_test[i])

plt.subplot(1, 3, 2)

plt.title("Ground Truth Mask")

plt.imshow(y\_test\_seg[i].squeeze(), cmap='gray')

plt.subplot(1, 3, 3)

plt.title("Predicted Mask")

plt.imshow(pred[i - 6].squeeze(), cmap='gray') # Adjust indexing for prediction

plt.show()