def build\_unet(input\_shape, n\_classes):

    def conv\_block(inputs, num\_filters):

        x = Conv2D(num\_filters, 3, activation='relu', padding='same')(inputs)

        x = Conv2D(num\_filters, 3, activation='relu', padding='same')(x)

        x = BatchNormalization()(x)

        x = tf.keras.layers.LeakyReLU(negative\_slope=0.2)(x)  # LeakyReLU kullanımı

        x = tf.keras.layers.Dropout(0.2)(x)  # Dropout eklenmesi

        return x

    def encoder\_block(inputs, num\_filters):

        x = conv\_block(inputs, num\_filters)

        p = MaxPool2D((2, 2))(x)

        return x, p

    def decoder\_block(inputs, skip, num\_filters):

        x = Conv2DTranspose(num\_filters, (3, 3), strides=(2, 2), padding='same')(inputs)

        x = tf.image.resize(x, tf.shape(skip)[1:3], method='nearest')  # Conv2DTranspose çıktısını skip ile boyutlandırma

        x = Concatenate()([x, skip])

        x = conv\_block(x, num\_filters)

        return x

    inputs = tf.keras.layers.Input(shape=input\_shape)

    # ENCODER

    s1, p1 = encoder\_block(inputs, 64)

    s2, p2 = encoder\_block(p1, 128)

    s3, p3 = encoder\_block(p2, 256)

    s4, p4 = encoder\_block(p3, 512)

    # BRIDGE

    b1 = conv\_block(p4, 1024)

    # DECODER

    d1 = decoder\_block(b1, s4, 512)

    d2 = decoder\_block(d1, s3, 256)

    d3 = decoder\_block(d2, s2, 128)

    d4 = decoder\_block(d3, s1, 64)

    # Additional Convolutional Blocks for Deepening the Network

    d5 = conv\_block(d4, 64)

    d6 = conv\_block(d5, 64)

    # Çıkış katmanını tanımlayalım

    if n\_classes == 1:

        activation = 'sigmoid'

    else:

        activation = 'softmax'

    outputs = Conv2D(n\_classes, 1, padding="same", activation=activation)(d6)

    model = tf.keras.Model(inputs, outputs, name="U-Net")

    return model