```
from IPython.display import clear_output
!pip install tf_explain
clear_output()

import os
import keras
import numpy as np
import pandas as pd
from glob import glob
import tensorflow as tf
import tensorflow.image as tfi

from tensorflow.keras.utils import load_img, img_to_array
from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt

from keras.models import Model
from keras.layers import Layer
from keras.layers import Conv2D
from keras.layers import Dropout
from keras.layers import UpSampling2D
from keras.layers import concatenate
from keras.layers import Add
from keras.layers import Multiply
from keras.layers import Input
from keras.layers import MaxPool2D
from keras.layers import BatchNormalization

from keras.callbacks import Callback
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from tf_explain.core.grad_cam import GradCAM

from keras.metrics import MeanIoU
```

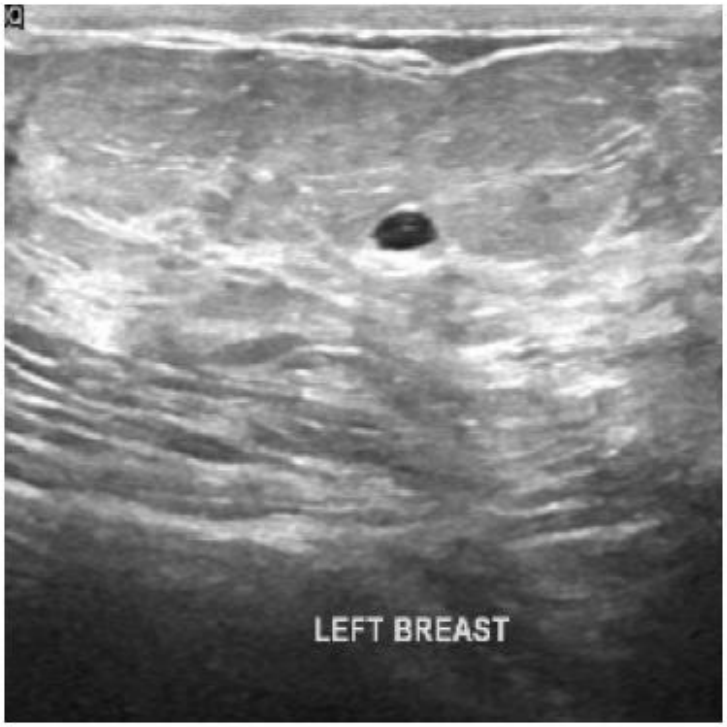
Data

```
def load_image(image, SIZE):
    return np.round(tfi.resize(img_to_array(load_img(image))/255.,(SIZE,
SIZE)),4)

def load_images(image_paths, SIZE, mask=False, trim=None):
    if trim is not None:
        image_paths = image_paths[:trim]

    if mask:
        images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 1))
    else:
        images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 3))
```

```
    for i,image in enumerate(image_paths):
        img = load_image(image,SIZE)
        if mask:
            images[i] = img[:,:,:1]
        else:
            images[i] = img

    return images

def show_image(image, title=None, cmap=None, alpha=1):
    plt.imshow(image, cmap=cmap, alpha=alpha)
    if title is not None:
        plt.title(title)
    plt.axis('off')

def show_mask(image, mask, cmap=None, alpha=0.4):
    plt.imshow(image)
    plt.imshow(tf.squeeze(mask), cmap=cmap, alpha=alpha)
    plt.axis('off')

SIZE = 256

root_path = '../input/breast-ultrasound-images-dataset/Dataset_BUSI_with_GT/'
classes = sorted(os.listdir(root_path))
classes
```

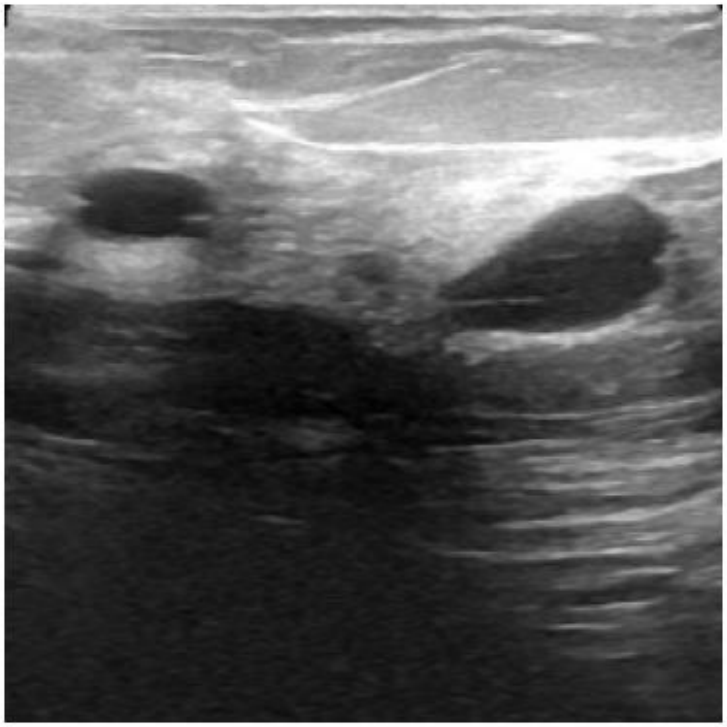
```
['benign', 'malignant', 'normal']
```

```
single_mask_paths = sorted([sorted(glob(root_path + name + "/*mask.png")) for
name in classes])
double_mask_paths = sorted([sorted(glob(root_path + name + "/*mask_1.png"))
for name in classes])

image_paths = []
mask_paths = []
for class_path in single_mask_paths:
    for path in class_path:
        img_path = path.replace('_mask','')
        image_paths.append(img_path)
        mask_paths.append(path)

show_image(load_image(image_paths[0], SIZE))
```

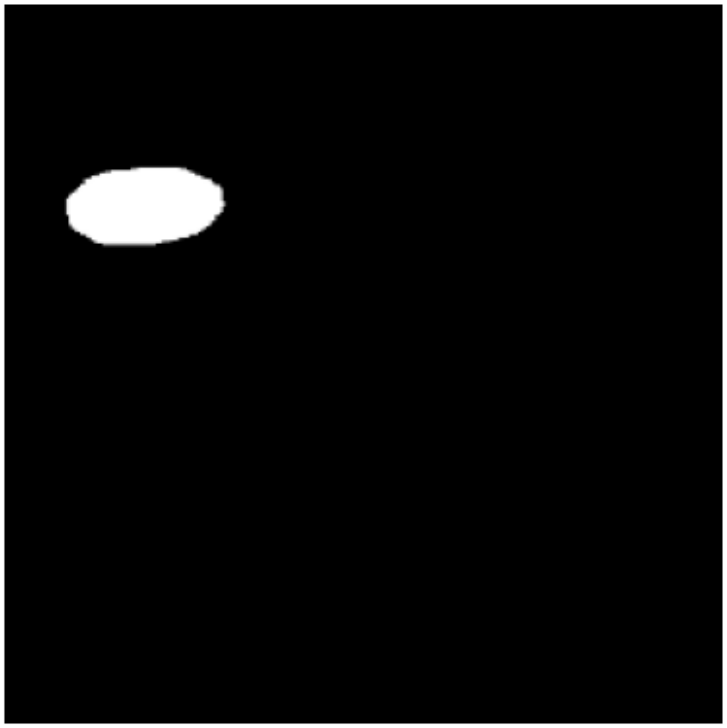



```
show_mask(load_image(image_paths[0], SIZE), load_image(mask_paths[0],
SIZE)[:,:,0], alpha=0.6)
```


```
show_image(load_image('../input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/benign/benign (100).png', SIZE))
```



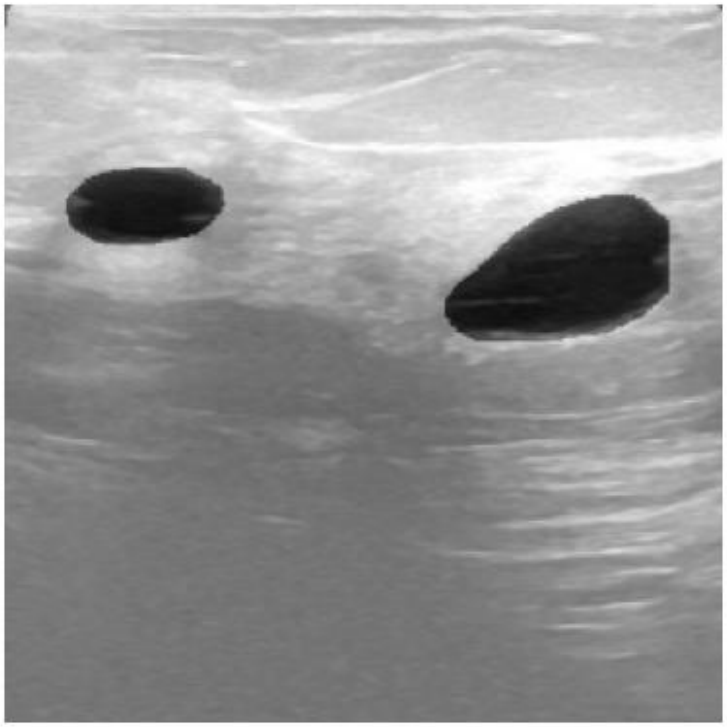
```
show_image(load_image('../input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/benign/benign (100)_mask_1.png', SIZE))
```



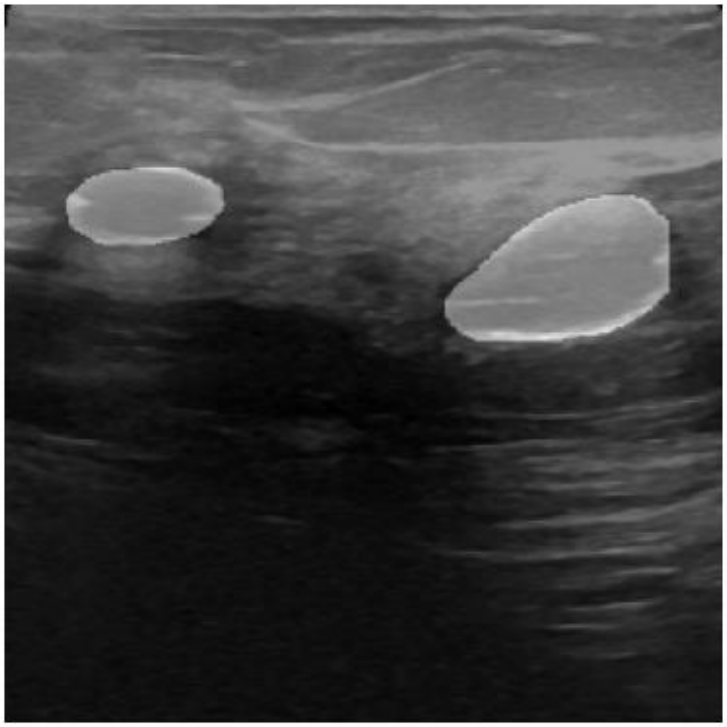
```
show_image(load_image('../input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/benign/benign (100)_mask.png', SIZE))
```

```
img = np.zeros((1,SIZE,SIZE,3))
mask1 = load_image('../input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/benign/benign (100)_mask_1.png', SIZE)
mask2 = load_image('../input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/benign/benign (100)_mask.png', SIZE)

img = img + mask1 + mask2
img = img[0,:,:,0]
show_image(img, cmap='gray')
```

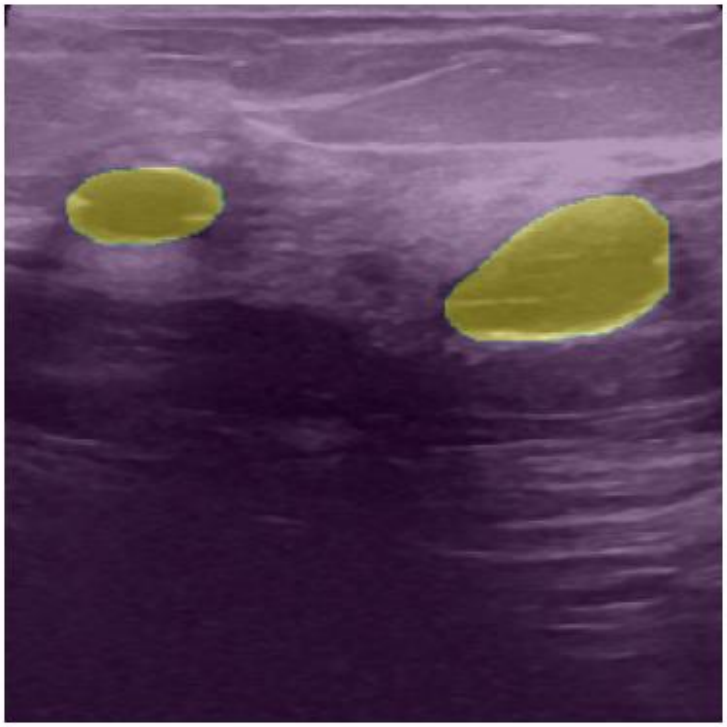
```
show_image(load_image('../input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/benign/benign (100).png', SIZE))
plt.imshow(img, cmap='binary', alpha=0.4)
plt.axis('off')
plt.show()
```



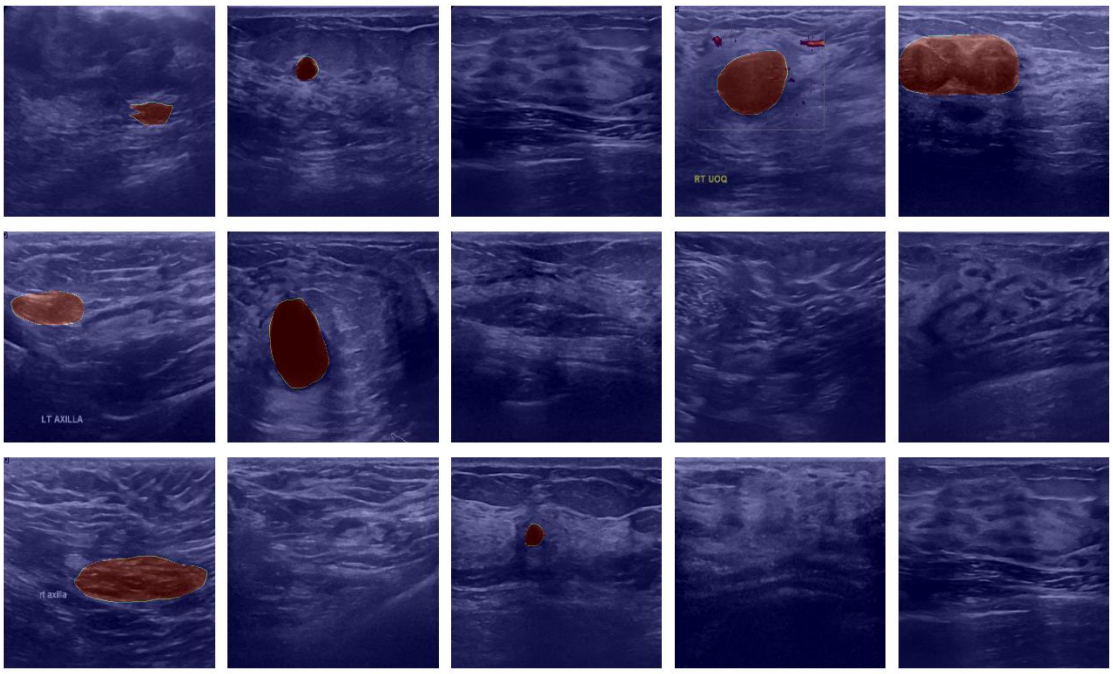
```
show_image(load_image('../input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/benign/benign (100).png', SIZE))
plt.imshow(img, cmap='gray', alpha=0.4)
plt.axis('off')
plt.show()
```



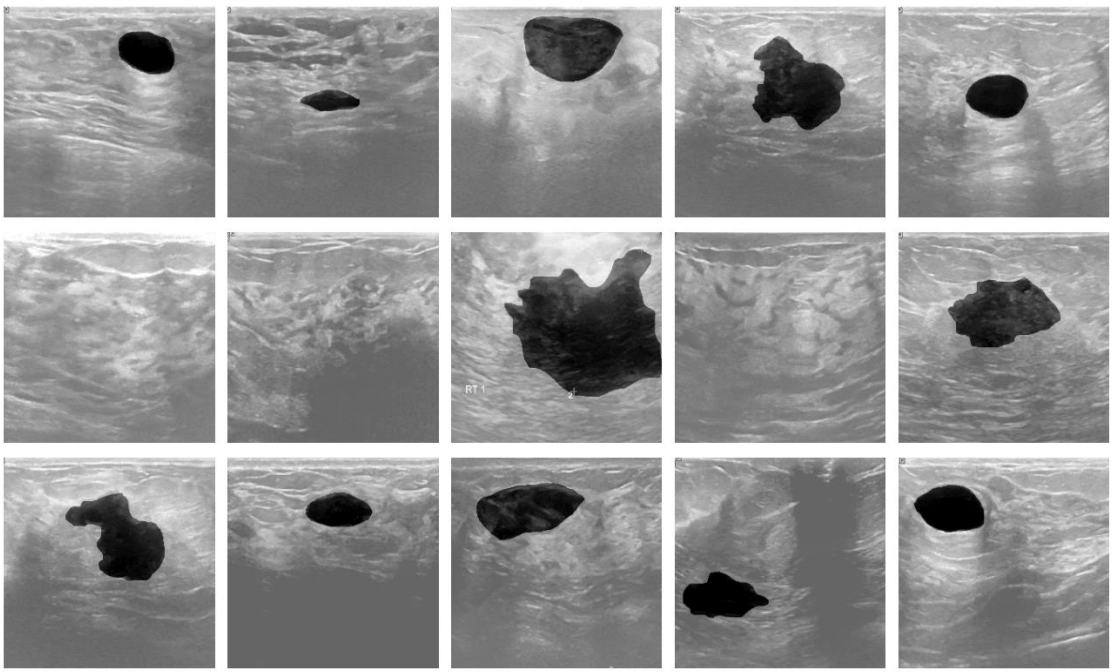
```
show_image(load_image('../input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/benign/benign (100).png', SIZE))
plt.imshow(img, alpha=0.4)
plt.axis('off')
plt.show()
```



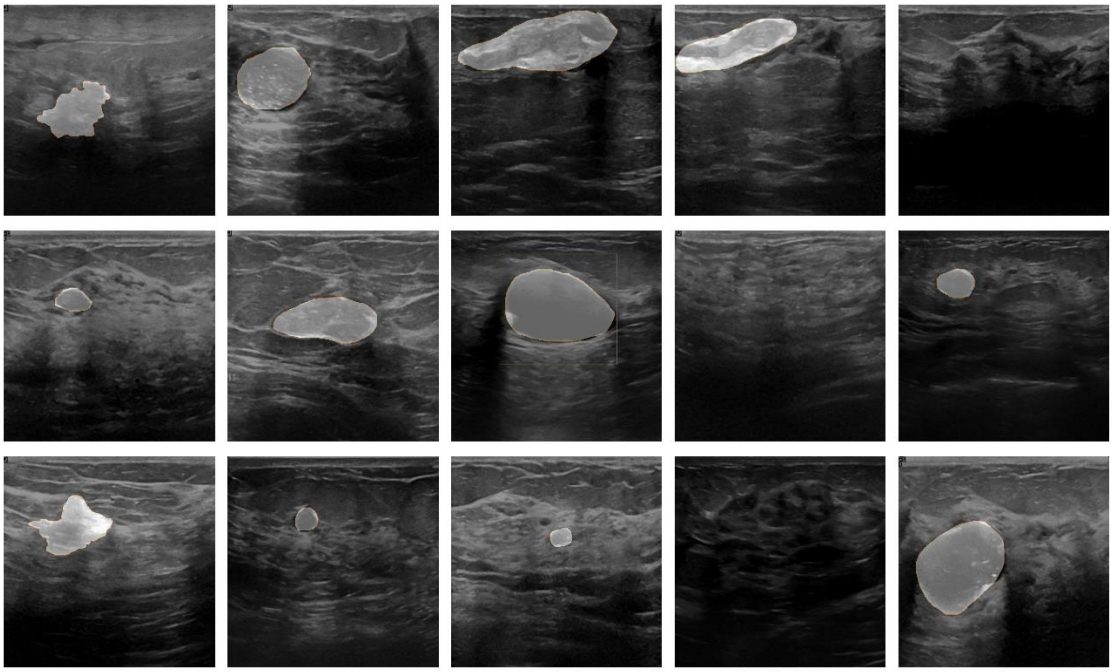
```
images = load_images(image_paths, SIZE)
masks = load_images(mask_paths, SIZE, mask=True)

plt.figure(figsize=(13,8))
for i in range(15):
    plt.subplot(3,5,i+1)
    id = np.random.randint(len(images))
    show_mask(images[id], masks[id], cmap='jet')
plt.tight_layout()
plt.show()
```



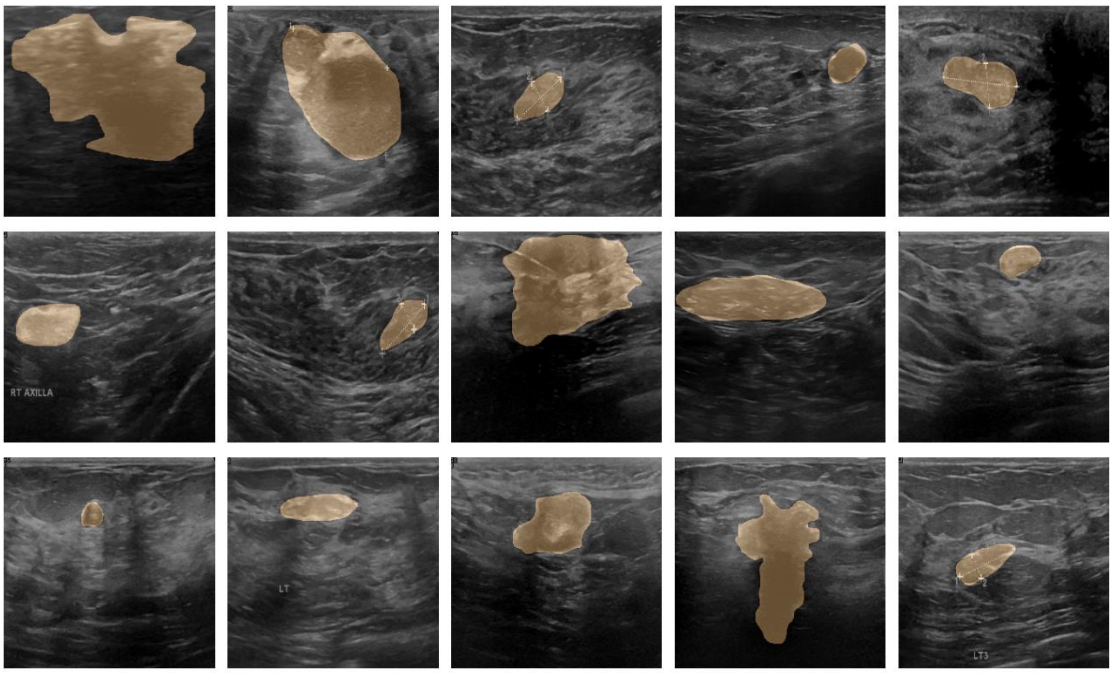

```
plt.figure(figsize=(13,8))
for i in range(15):
    plt.subplot(3,5,i+1)
    id = np.random.randint(len(images))
    show_mask(images[id], masks[id], cmap='binary')
plt.tight_layout()
plt.show()
```



```
plt.figure(figsize=(13,8))
for i in range(15):
    plt.subplot(3,5,i+1)
    id = np.random.randint(len(images))
    show_mask(images[id], masks[id], cmap='afmhot')
plt.tight_layout()
plt.show()
```



```
plt.figure(figsize=(13,8))
for i in range(15):
    plt.subplot(3,5,i+1)
    id = np.random.randint(len(images))
    show_mask(images[id], masks[id], cmap='copper')
plt.tight_layout()
plt.show()
```




```
import tensorflow as tf
from tensorflow.keras.layers import Layer, Conv2D, Dropout, MaxPool2D,
UpSampling2D, concatenate, BatchNormalization, Add, Multiply, Input,
Activation, GlobalAveragePooling2D, GlobalMaxPooling2D, Dense, Reshape
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import Callback, ModelCheckpoint
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.metrics import MeanIoU

class EncoderBlock(Layer):
    def __init__(self, filters, rate, pooling=True, **kwargs):
        super(EncoderBlock, self).__init__(**kwargs)
        self.filters = filters
        self.rate = rate
        self.pooling = pooling
        self.c1 = Conv2D(filters, kernel_size=3, padding='same',
activation='relu', kernel_initializer='he_normal')
        self.drop = Dropout(rate)
        self.c2 = Conv2D(filters, kernel_size=3, padding='same',
activation='relu', kernel_initializer='he_normal')
        self.pool = MaxPool2D()

    def call(self, X):
        x = self.c1(X)
        x = self.drop(x)
        x = self.c2(x)
        if self.pooling:
            y = self.pool(x)
```

```
            return y, x
        return x

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "filters": self.filters, "rate": self.rate,
"pooling": self.pooling}

class DecoderBlock(Layer):
    def __init__(self, filters, rate, **kwargs):
        super(DecoderBlock, self).__init__(**kwargs)
        self.filters = filters
        self.rate = rate
        self.up = UpSampling2D()
        self.net = EncoderBlock(filters, rate, pooling=False)

    def build(self, input_shape):
        X_prev_shape = input_shape[0]
        self.attention = CBAMAttentionGate(filters=X_prev_shape[-1])
        super().build(input_shape)

    def call(self, X):
        X_prev, skip_X = X
        x = self.up(X_prev)
        a = self.attention([x, skip_X])
        c_ = concatenate([x, a])
        x = self.net(c_)
        return x

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "filters": self.filters, "rate": self.rate}

class CBAMAttentionGate(Layer):
    def __init__(self, filters, reduction_ratio=16, **kwargs):
        super(CBAMAttentionGate, self).__init__(**kwargs)
        self.filters = filters
        self.reduction_ratio = reduction_ratio

        self.avg_pool = GlobalAveragePooling2D()
        self.max_pool = GlobalMaxPooling2D()
        self.fc1 = Dense(filters // reduction_ratio, activation='relu',
kernel_initializer='he_normal')
        self.fc2 = Dense(filters, activation='sigmoid',
kernel_initializer='he_normal')

        self.conv_spatial = Conv2D(1, kernel_size=7, padding='same',
activation='sigmoid', kernel_initializer='he_normal')
```

```
        self.conv_skip = Conv2D(filters, kernel_size=3, padding='same',
activation='relu', kernel_initializer='he_normal')
        self.bn = BatchNormalization()

    def call(self, inputs):
        X, skip_X = inputs

        avg_pool = self.avg_pool(X)
        max_pool = self.max_pool(X)
        avg_out = self.fc2(self.fc1(avg_pool))
        max_out = self.fc2(self.fc1(max_pool))
        channel_attn = tf.sigmoid(avg_out + max_out)
        channel_attn = Reshape((1, 1, self.filters))(channel_attn)
        x = Multiply()([X, channel_attn])

        avg_spatial = tf.reduce_mean(x, axis=-1, keepdims=True)
        max_spatial = tf.reduce_max(x, axis=-1, keepdims=True)
        spatial_concat = concatenate([avg_spatial, max_spatial])
        spatial_attn = self.conv_spatial(spatial_concat)
        x = Multiply()([x, spatial_attn])

        skip = self.conv_skip(skip_X)
        if skip.shape[1] != x.shape[1] or skip.shape[2] != x.shape[2]:
            skip = tf.image.resize(skip, size=(x.shape[1], x.shape[2]))
        x = Add()([x, skip])
        x = self.bn(x)
        return x

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "filters": self.filters, "reduction_ratio":
self.reduction_ratio}

input_layer = Input(shape=images.shape[-3:])

p1, c1 = EncoderBlock(32, 0.1, name="Encoder1")(input_layer)
p2, c2 = EncoderBlock(64, 0.1, name="Encoder2")(p1)
p3, c3 = EncoderBlock(128, 0.2, name="Encoder3")(p2)
p4, c4 = EncoderBlock(256, 0.2, name="Encoder4")(p3)
encoding = EncoderBlock(512, 0.3, pooling=False, name="Encoding")(p4)

d1 = DecoderBlock(256, 0.2, name="Decoder1")([encoding, c4])
d2 = DecoderBlock(128, 0.2, name="Decoder2")([d1, c3])
d3 = DecoderBlock(64, 0.1, name="Decoder3")([d2, c2])
d4 = DecoderBlock(32, 0.1, name="Decoder4")([d3, c1])

output_layer = Conv2D(1, kernel_size=1, activation='sigmoid',
padding='same')(d4)
```

```
model = Model(inputs=[input_layer], outputs=[output_layer])

model.summary()

Model: "model_1"
_________________________________________________________________________________
_____________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================
=====================
 input_8 (InputLayer)        [(None, 256, 256, 3  0           []
                             )]

 Encoder1 (EncoderBlock)     ((None, 128, 128, 3  10144
['input_8[0][0]']
                             2),
                              (None, 256, 256, 3
                             2))

 Encoder2 (EncoderBlock)     ((None, 64, 64, 64)  55424
['Encoder1[0][0]']
                             , (None, 128, 128,
                             64))

 Encoder3 (EncoderBlock)     ((None, 32, 32, 128  221440
['Encoder2[0][0]']
                             ),
                              (None, 64, 64, 128
                             ))

 Encoder4 (EncoderBlock)     ((None, 16, 16, 256  885248
['Encoder3[0][0]']
                             ),
                              (None, 32, 32, 256
                             ))

 Encoding (EncoderBlock)     (None, 16, 16, 512)  3539968
['Encoder4[0][0]']

 Decoder1 (DecoderBlock)     (None, 32, 32, 256)  4165251
['Encoding[0][0]',

'Encoder4[0][1]']

 Decoder2 (DecoderBlock)     (None, 64, 64, 128)  1042291
['Decoder1[0][0]',

'Encoder3[0][1]']
```

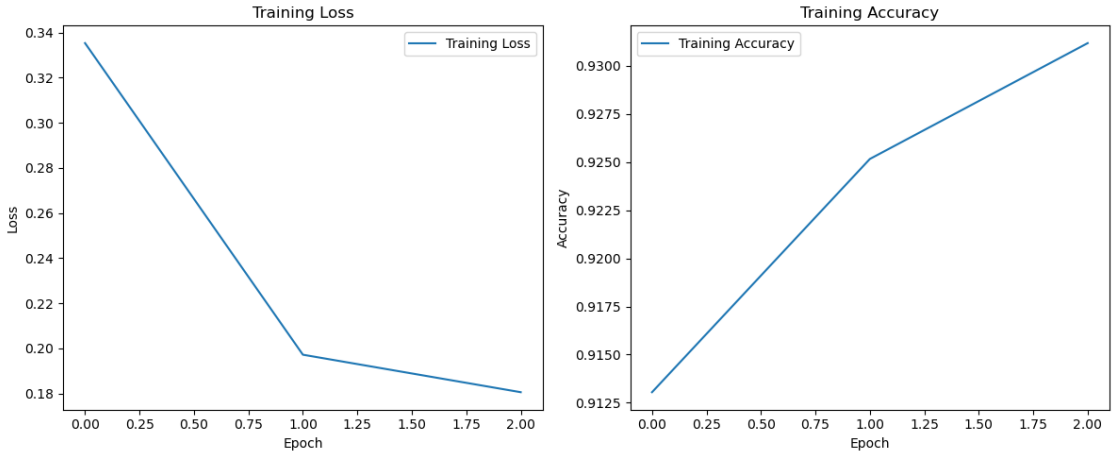
```
 Decoder3 (DecoderBlock)        (None, 128, 128, 64  261099
['Decoder2[0][0]',
                                )
'Encoder2[0][1]']

 Decoder4 (DecoderBlock)        (None, 256, 256, 32  65575
['Decoder3[0][0]',
                                )
'Encoder1[0][1]']

 conv2d_113 (Conv2D)            (None, 256, 256, 1)  33
['Decoder4[0][0]']

==================================================================================
Total params: 10,246,473
Trainable params: 10,244,553
Non-trainable params: 1,920
__________________________________________________________________________________
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
history = model.fit(images, masks, epochs=3, batch_size=16)
```

```
Epoch 1/3

2025-05-17 08:55:27.607123: E
tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed:
INVALID_ARGUMENT: Size of values 0 does not match size of permutation 4 @
fanin shape inmodel_1/Encoder1/dropout_33/dropout/SelectV2-2-
TransposeNHWCToNCHW-LayoutOptimizer

49/49 [==============================] - 44s 438ms/step - loss: 0.3353 -
accuracy: 0.9130
Epoch 2/3
49/49 [==============================] - 18s 367ms/step - loss: 0.1972 -
accuracy: 0.9252
Epoch 3/3
49/49 [==============================] - 18s 367ms/step - loss: 0.1806 -
accuracy: 0.9312
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss')
```

```
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.title('Training Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(20,25))
n=0
for i in range(1,(5*3)+1):
    plt.subplot(5,3,i)
    if n==0:
        id = np.random.randint(len(images))
        image = images[id]
        mask = masks[id]
        pred_mask = model.predict(image[np.newaxis,...])

        plt.title("Original Mask")
        show_mask(image, mask)
        n+=1
    elif n==1:
        plt.title("Predicted Mask")
        show_mask(image, pred_mask)
        n+=1
    elif n==2:
        pred_mask = (pred_mask>0.5).astype('float')
        plt.title("Processed Mask")
        show_mask(image, pred_mask)
        n=0
```

```
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 1s 938ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
```