# XAI - DA

## Implementing LRP on Breast Cancer Segmentation problem

### Requirements

```
ipykernel
opencv-python
numpy
pandas
innvestigate
matplotlib
scikit-image
tensorflow
keras
tensorflow_datasets
```

**Note:** Enable eager execution before starting the program using

```
tf.compat.v1.enable_eager_execution()
```

This setting will be disabled right before implementing the XAI technique

### Importing data

```
img_size = (256,256)
num_classes = 2
batch_size = 16
```

Function to create a dataset array from the images

```
def createDataset(Images, Masks):
    Images = filter(lambda img: img.endswith(".tif"), Images)
    Masks = filter(lambda img: img.endswith(".TIF"), Masks)
#     print(list(Masks))
    Images = map(lambda img: os.path.join(ImagesDir, img), Images)
    Masks = map(lambda imag: os.path.join(MasksDir, imag), Masks)
#     print(list(Masks))
    Images = sorted(Images)
    Masks = sorted(Masks)

    return Images, Masks

dataset = createDataset(os.listdir(ImagesDir), os.listdir(MasksDir))
```

Converting to a Pandas DF

```
dataset = pd.DataFrame(dataset).T
```

```
class CancerDataset(keras.utils.Sequence):
    """Helper to iterate over the data (as Numpy arrays)."""

    def __init__(self, batch_size, img_size, input_img_paths, target_img_paths):
        self.batch_size = batch_size
        self.img_size = img_size
        self.input_img_paths = input_img_paths
        self.target_img_paths = target_img_paths

    def __len__(self):
        return len(self.target_img_paths) // self.batch_size

    def __getitem__(self, idx):
        """Returns tuple (input, target) correspond to batch #idx."""
        i = idx * self.batch_size
        batch_input_img_paths = self.input_img_paths[i : i + self.batch_size]
        batch_target_img_paths = self.target_img_paths[i : i + self.batch_size]
```

```
        x = np.zeros((self.batch_size,) + self.img_size + (3,), dtype="float32")
        for j, path in enumerate(batch_input_img_paths):
            image = imread(path)
#             print(image.shape)
#             imshow(image)
#             image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = cv2.resize(image, img_size)
            image = image/255.0
            x[j] = image
        y = np.zeros((self.batch_size,) + self.img_size + (1,), dtype="float32")
        for j, path in enumerate(batch_target_img_paths):
            image = imread(path)
#             //print(image.shape)
#             imshow(image)
#             image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image = cv2.resize(image, img_size)

#             print(image.shape)
#             imshow(image)
#             image = keras.utils.to_categorical(image, axis = -1)
            image = np.expand_dims(image, axis = -1)
            image = image/255.0
            y[j] = image
        return x, y
```

**Function to create the model**

```
def get_model(weights = None, flatten = False, last_activation="sigmoid"):
    inputs = Input((256, 256, 3))
    # Change integer to float and also scale pixel values

    # Contraction/Encoder path
    # Block 1
    c1 = Conv2D(filters=16, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(inputs)
    c1 = Dropout(0.1)(c1)
    c1 = Conv2D(filters=16, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(c1)
    p1 = MaxPooling2D(pool_size=(2,2))(c1)
    # Block 2
    c2 = Conv2D(filters=32, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv2D(filters=32, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(c2)
    p2 = MaxPooling2D(pool_size=(2,2))(c2)
    # Block 3
    c3 = Conv2D(filters=64, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv2D(filters=64, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(c3)
    p3 = MaxPooling2D(pool_size=(2,2))(c3)
    # Block 4
    c4 = Conv2D(filters=128, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv2D(filters=128, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(c4)
    p4 = MaxPooling2D(pool_size=(2,2))(c4)
    # Block 5
    c5 = Conv2D(filters=256, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
                              padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv2D(filters=256, kernel_size=(3,3),
                               activation='relu', kernel_initializer='he_normal',
```

```
                             padding='same')(c5)

    # Expansion/Decoder path
    # Block 6
    u6 = Conv2DTranspose(filters=128, kernel_size=(2,2), strides = (2,2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(filters=128, kernel_size=(3,3),
                              activation='relu', kernel_initializer='he_normal',
                             padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv2D(filters=128, kernel_size=(3,3),
                              activation='relu', kernel_initializer='he_normal',
                             padding='same')(c6)

    # Block 7
    u7 = Conv2DTranspose(filters=64, kernel_size=(2,2), strides = (2,2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(filters=64, kernel_size=(3,3),
                              activation='relu', kernel_initializer='he_normal',
                             padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv2D(filters=64, kernel_size=(3,3),
                              activation='relu', kernel_initializer='he_normal',
                             padding='same')(c7)

    # Block 8
    u8 = Conv2DTranspose(filters=32, kernel_size=(2,2), strides = (2,2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(filters=32, kernel_size=(3,3),
                              activation='relu', kernel_initializer='he_normal',
                             padding='same')(u8)
    c8 = Dropout(0.1)(c8)
    c8 = Conv2D(filters=32, kernel_size=(3,3),
                              activation='relu', kernel_initializer='he_normal',
                             padding='same')(c8)

    # Block 9
    u9 = Conv2DTranspose(filters=16, kernel_size=(2,2), strides = (2,2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv2D(filters=16, kernel_size=(3,3),
                              activation='relu', kernel_initializer='he_normal',
                             padding='same')(u9)
    c9 = Dropout(0.1)(c9)
    c9 = Conv2D(filters=16, kernel_size=(3,3),
                              activation='relu', kernel_initializer='he_normal',
                             padding='same')(c9)
    # Outputs
    outputs = Conv2D(filters=1, kernel_size=(1,1),
                              activation=last_activation)(c9)

    if(flatten):
        outputs = Flatten()(outputs)

    model = Model(inputs=[inputs], outputs=[outputs])

    if weights is not None:
        model.load_weights(weights)
    return model

model = get_model()

model.summary()
```

### Model summary

```
Model: "model_3"
_________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_4 (InputLayer)        [(None, 256, 256, 3)]        0         []

 conv2d_57 (Conv2D)          (None, 256, 256, 16)         448       ['input_4[0][0]']

 dropout_27 (Dropout)        (None, 256, 256, 16)         0         ['conv2d_57[0][0]']

 conv2d_58 (Conv2D)          (None, 256, 256, 16)         2320      ['dropout_27[0][0]']
```

```
max_pooling2d_12 (MaxPooli  (None, 128, 128, 16)         0         ['conv2d_58[0][0]']
ng2D)

conv2d_59 (Conv2D)          (None, 128, 128, 32)         4640      ['max_pooling2d_12[0][0]']

dropout_28 (Dropout)        (None, 128, 128, 32)         0         ['conv2d_59[0][0]']

conv2d_60 (Conv2D)          (None, 128, 128, 32)         9248      ['dropout_28[0][0]']

max_pooling2d_13 (MaxPooli  (None, 64, 64, 32)           0         ['conv2d_60[0][0]']
ng2D)

conv2d_61 (Conv2D)          (None, 64, 64, 64)           18496     ['max_pooling2d_13[0][0]']

dropout_29 (Dropout)        (None, 64, 64, 64)           0         ['conv2d_61[0][0]']

conv2d_62 (Conv2D)          (None, 64, 64, 64)           36928     ['dropout_29[0][0]']

max_pooling2d_14 (MaxPooli  (None, 32, 32, 64)           0         ['conv2d_62[0][0]']
ng2D)

conv2d_63 (Conv2D)          (None, 32, 32, 128)          73856     ['max_pooling2d_14[0][0]']

dropout_30 (Dropout)        (None, 32, 32, 128)          0         ['conv2d_63[0][0]']

conv2d_64 (Conv2D)          (None, 32, 32, 128)          147584    ['dropout_30[0][0]']

max_pooling2d_15 (MaxPooli  (None, 16, 16, 128)          0         ['conv2d_64[0][0]']
ng2D)

conv2d_65 (Conv2D)          (None, 16, 16, 256)          295168    ['max_pooling2d_15[0][0]']

dropout_31 (Dropout)        (None, 16, 16, 256)          0         ['conv2d_65[0][0]']

conv2d_66 (Conv2D)          (None, 16, 16, 256)          590080    ['dropout_31[0][0]']

conv2d_transpose_12 (Conv2  (None, 32, 32, 128)          131200    ['conv2d_66[0][0]']
DTranspose)

concatenate_12 (Concatenat  (None, 32, 32, 256)          0         ['conv2d_transpose_12[0][0]',
e)                                                                  'conv2d_64[0][0]']

conv2d_67 (Conv2D)          (None, 32, 32, 128)          295040    ['concatenate_12[0][0]']

dropout_32 (Dropout)        (None, 32, 32, 128)          0         ['conv2d_67[0][0]']

conv2d_68 (Conv2D)          (None, 32, 32, 128)          147584    ['dropout_32[0][0]']

conv2d_transpose_13 (Conv2  (None, 64, 64, 64)           32832     ['conv2d_68[0][0]']
DTranspose)

concatenate_13 (Concatenat  (None, 64, 64, 128)          0         ['conv2d_transpose_13[0][0]',
e)                                                                  'conv2d_62[0][0]']

conv2d_69 (Conv2D)          (None, 64, 64, 64)           73792     ['concatenate_13[0][0]']

dropout_33 (Dropout)        (None, 64, 64, 64)           0         ['conv2d_69[0][0]']

conv2d_70 (Conv2D)          (None, 64, 64, 64)           36928     ['dropout_33[0][0]']

conv2d_transpose_14 (Conv2  (None, 128, 128, 32)         8224      ['conv2d_70[0][0]']
DTranspose)

concatenate_14 (Concatenat  (None, 128, 128, 64)         0         ['conv2d_transpose_14[0][0]',
e)                                                                  'conv2d_60[0][0]']

conv2d_71 (Conv2D)          (None, 128, 128, 32)         18464     ['concatenate_14[0][0]']

dropout_34 (Dropout)        (None, 128, 128, 32)         0         ['conv2d_71[0][0]']

conv2d_72 (Conv2D)          (None, 128, 128, 32)         9248      ['dropout_34[0][0]']

conv2d_transpose_15 (Conv2  (None, 256, 256, 16)         2064      ['conv2d_72[0][0]']
DTranspose)

concatenate_15 (Concatenat  (None, 256, 256, 32)         0         ['conv2d_transpose_15[0][0]',
e)                                                                  'conv2d_58[0][0]']
```

```
 conv2d_73 (Conv2D)          (None, 256, 256, 16)         4624      ['concatenate_15[0][0]']

 dropout_35 (Dropout)        (None, 256, 256, 16)         0         ['conv2d_73[0][0]']

 conv2d_74 (Conv2D)          (None, 256, 256, 16)         2320      ['dropout_35[0][0]']

 conv2d_75 (Conv2D)          (None, 256, 256, 1)          17        ['conv2d_74[0][0]']

==================================================================================================
Total params: 1941105 (7.40 MB)
Trainable params: 1941105 (7.40 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

#### Splitting the data 85% for training and 15% for validation

```
val_samples = 6 # 85% Training -- 15% Validation
random.Random(1822).shuffle(imageList)
random.Random(1822).shuffle(maskList)
print(imageList[0])
print(maskList[0])
train_input_img_paths = imageList[:-val_samples]
train_target_img_paths = maskList[:-val_samples]
print(train_input_img_paths[0], train_target_img_paths[0])
val_input_img_paths = imageList[-val_samples:]
val_target_img_paths = maskList[-val_samples:]

# Instantiate data Sequences for each split
train_gen = CancerDataset(
    batch_size, img_size, train_input_img_paths, train_target_img_paths
)
val_gen = CancerDataset(6, img_size, val_input_img_paths, val_target_img_paths)
```

#### Loss function

```
# Loss function
def bce_loss(y_true,y_pred):
    y_true=K.cast(y_true,'float32')
    y_pred=K.cast(y_pred,'float32')
    return tf.keras.losses.binary_crossentropy(y_true,y_pred)
def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)

    y_true_f = K.cast(y_true_f, 'float32')
    y_pred_f = K.cast(y_pred_f, 'float32')

    intersection = K.sum(y_true_f * y_pred_f)
    dice_coef_v = (2. * intersection + 1.) / (K.sum(y_true_f) + K.sum(y_pred_f) + 1.)
    return dice_coef_v
def dice_loss(y_true, y_pred):
    dice_loss_v = 1 - dice_coef(y_true, y_pred)
    return dice_loss_v
def bce_dice_loss(y_true, y_pred):
    bce_dice_loss_v = bce_loss(y_true, y_pred) + dice_loss(y_true, y_pred)
    return bce_dice_loss_v
```

#### Running the model itself

```
optimizer = keras.optimizers.SGD(learning_rate=0.1)
model.compile(optimizer="adam", loss=bce_dice_loss, metrics=[dice_coef])

callbacks = [
    keras.callbacks.ModelCheckpoint("cancerSegmentation.h5", save_best_only=True)
]

epochs = 300
modelunet=model.fit(train_gen, epochs=epochs, validation_data=val_gen, callbacks=callbacks)
```

#### Saving the model file

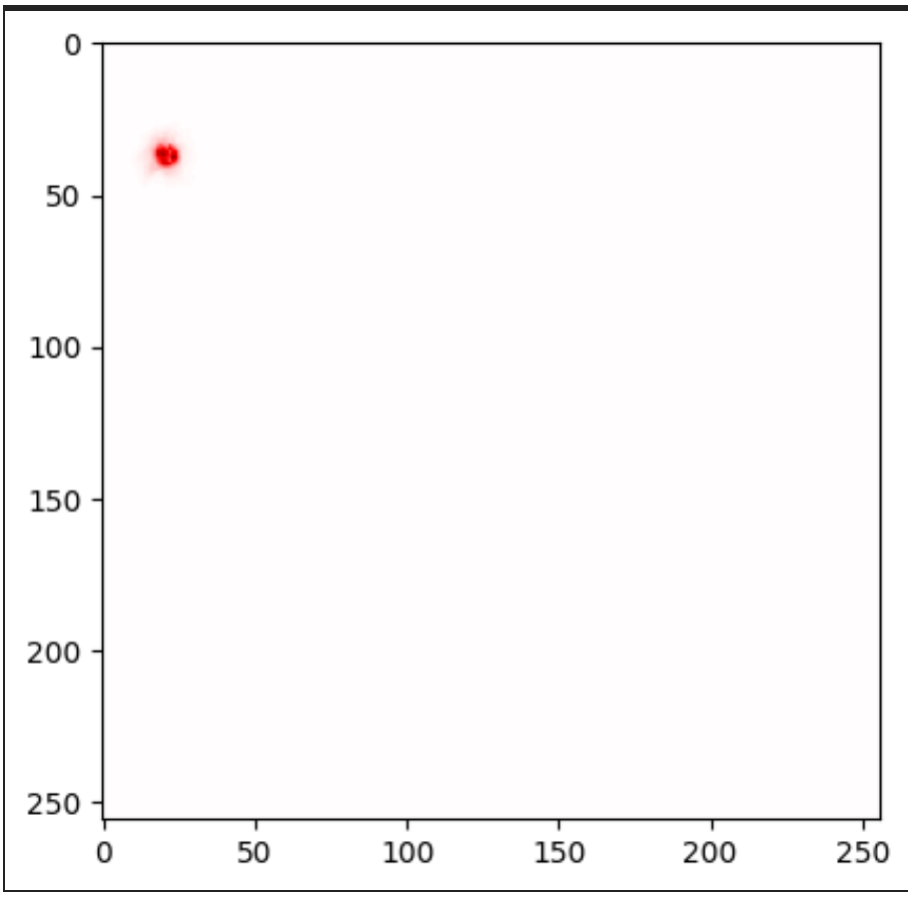
```
model.save("unet_breast_cancer.h5")
model.save_weights("unet_breast_cancer.weights.h5")
```

#### To load the model later on

```
model = tf.keras.models.load_model("unet_breast_cancer.h5", custom_objects={'bce_dice_loss': bce_dice_loss, 'dice_coef': dic
```

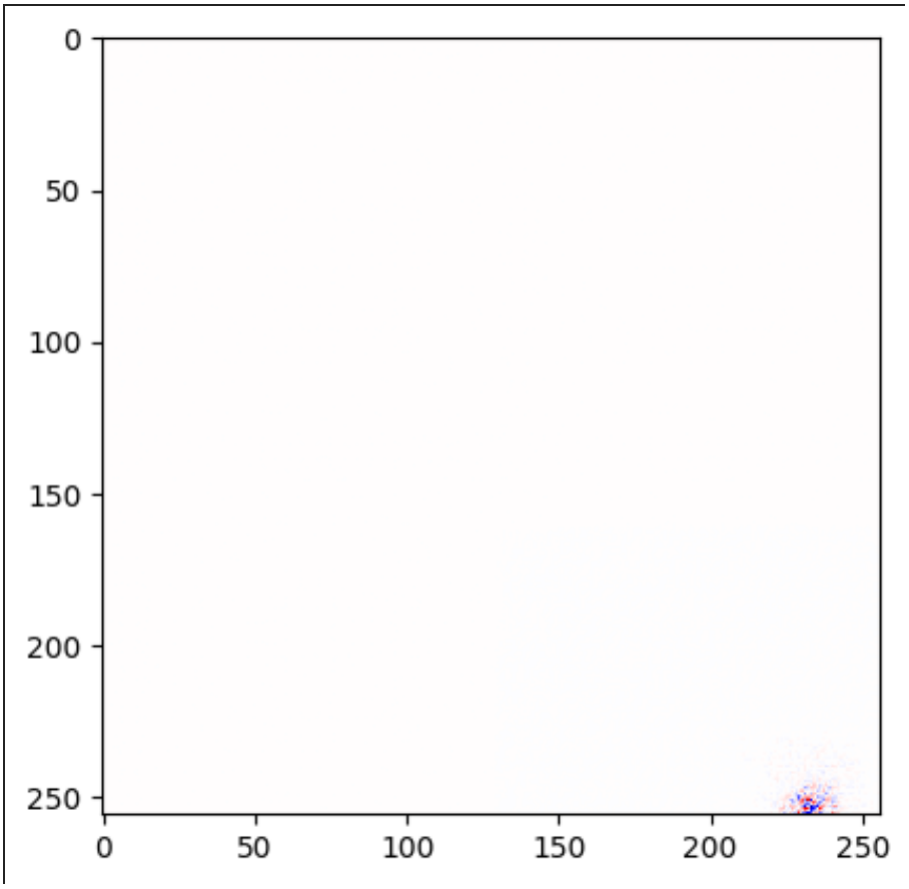
#### Running the model through Deep Taylor method with output

```
# modelWithoutSigmoid = inv.model_wo_output_activation(model)
tf.compat.v1.disable_eager_execution()
modelWithFlatten = get_model("unet_breast_cancer.weights.h5", flatten=True, last_activation="linear")
analyser = inv.create_analyzer("deep_taylor", modelWithFlatten)
x = np.expand_dims(val_gen.__getitem__(0)[0][2], axis = 0)

a = analyser.analyze(x)
a = a.sum(axis=np.argmax(np.asarray(a.shape) == 3))
a /= np.max(np.abs(a))
plt.imshow(a[0], cmap="seismic", clim=(-1, 1))
```



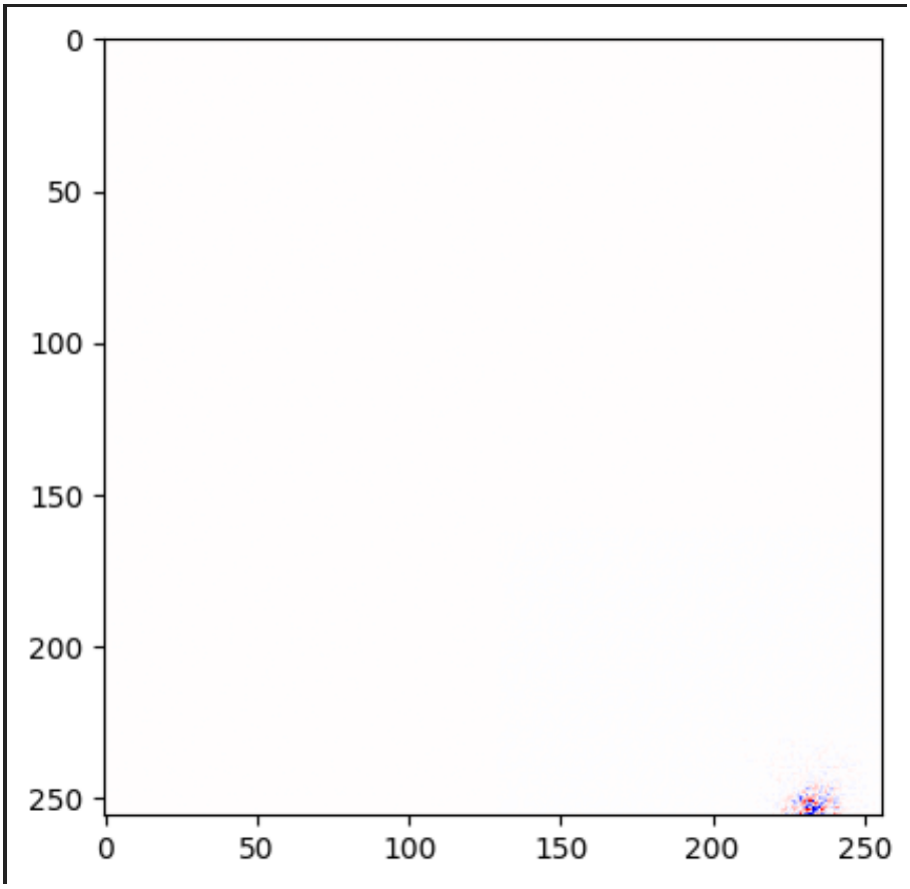

#### Running the model through LRP (Epsilon) method

```
analyser = inv.create_analyzer("lrp.epsilon", modelWithFlatten)
x = np.expand_dims(val_gen.__getitem__(0)[0][0], axis = 0)

a = analyser.analyze(x)
a = a.sum(axis=np.argmax(np.asarray(a.shape) == 3))
a /= np.max(np.abs(a))
plt.imshow(a[0], cmap="seismic", clim=(-1, 1))
```




**Running the model through LRP (Z) method**

```
analyser = inv.create_analyzer("lrp.z", modelWithFlatten)
x = np.expand_dims(val_gen.__getitem__(0)[0][0], axis = 0)

a = analyser.analyze(x)
a = a.sum(axis=np.argmax(np.asarray(a.shape) == 3))
a /= np.max(np.abs(a))
plt.imshow(a[0], cmap="seismic", clim=(-1, 1))
```


0
50
100
150
200
250
0
50
100
150
200
250