Breast Cancer Detection using Fast.ai Model

The goal for this task is to predict the presence or absence of cancer in mammography images.

In [1]:
```
ls ../input/rsna-breast-cancer-detection/train_images | head -1 -n 4
```

```
10006/
10011/
10025/
10038/
ls: write error
```

Each directory contains several images.

In [2]:
```
ls ../input/rsna-breast-cancer-detection/train_images/57175
```

```
1084689895.dcm  110562898.dcm  1564996746.dcm  269811751.dcm
```

In [3]:
```
import pandas as pd
from matplotlib import pyplot as plt
```

For each patient, we are to predict the presence or absence of cancer in the left and right breast.

Here is the result format.

In [4]:
```
sample_result
```

Out[4]:

	prediction_id	cancer
0	10008_L	0.021168
1	10008_R	0.021168

There are 11913 patients in the train set.

In [5]:
```
!ls -l ../input/rsna-breast-cancer-detection/train_images | wc -l # wc -l outputs one
```

```
11914
```

In test, we have access to just a single example.

In [6]:
```
ls ../input/rsna-breast-cancer-detection/test_images/10008
```

```
1591370361.dcm  361203119.dcm  68070693.dcm  736471439.dcm
```

The images we will work with are in the DICOM format.

DICOM image format overview

A DICOM file consists of a header and image data sets packed into a single file. The information within the header is organized as a constant and standardized series of tags. By extracting data

from these tags, one can access important information regarding patient demographics, study parameters, etc.

Let's examine a couple of images to gain a better intuition about the types of images with which we are working.

In [7]:
```
import pydicom
import numpy as np
from pydicom.pixel_data_handlers.util import apply_voi_lut
import matplotlib.pyplot as plt
import seaborn as sns
import os
from pathlib import Path
import glob
```

First, let's begin with the metadata. Each image contains rich metadata that might guide us in how we split the data for training. We might also want to directly provide some of this information to our model.

In [8]:
```
example = '../input/rsna-breast-cancer-detection/train_images/10006/1459541791.dcm'
pydicom.dcmread(example)
```

Out[8]:
```
Dataset.file_meta -------------------------------
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Digital X-Ray Image Storage - Fo
r Presentation
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.840.10009.1.2.3.10006.1.1459
541791
(0002, 0010) Transfer Syntax UID                 UI: JPEG 2000 Image Compression (Los
sless Only)
(0002, 0012) Implementation Class UID            UI: 1.2.840.113654.2.3.1995.2.12.0
(0002, 0013) Implementation Version Name         SH: 'PYDICOM 2.3.0'
-------------------------------------------------
(0008, 0018) SOP Instance UID                    UI: 1.2.840.10009.1.2.3.10006.1.1459
541791
(0008, 0023) Content Date                        DA: '20221118'
(0008, 0033) Content Time                        TM: '183901.792591'
(0010, 0020) Patient ID                          LO: '10006'
(0020, 000d) Study Instance UID                  UI: 1.2.840.10009.1.2.3.10006
(0020, 000e) Series Instance UID                 UI: 1.2.840.10009.1.2.3.10006.1
(0020, 0013) Instance Number                     IS: '1459541791'
(0020, 0062) Image Laterality                    CS: 'L'
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME1'
(0028, 0010) Rows                                US: 5355
(0028, 0011) Columns                             US: 4915
(0028, 0100) Bits Allocated                      US: 16
(0028, 0101) Bits Stored                         US: 16
(0028, 0102) High Bit                            US: 15
(0028, 0103) Pixel Representation                US: 0
(0028, 0120) Pixel Padding Value                 US: 3044
(0028, 1040) Pixel Intensity Relationship        CS: 'LOG'
(0028, 1041) Pixel Intensity Relationship Sign   SS: 1
(0028, 1050) Window Center                       DS: [1802.310000, 1802.310000, 2020.
704000, 1583.916000]
(0028, 1051) Window Width                        DS: [1091.970000, 1091.970000, 1091.
970000, 1091.970000]
(0028, 1052) Rescale Intercept                   DS: '0.0'
(0028, 1053) Rescale Slope                       DS: '1.0'
(0028, 1054) Rescale Type                        LO: 'US'
(0028, 1056) VOI LUT Function                    CS: 'SIGMOID'
(0028, 1350) Partial View                        CS: 'NO'
(0028, 2110) Lossy Image Compression             CS: '00'
(7fe0, 0010) Pixel Data                          OW: Array of 4362044 elements
```

look at some images.

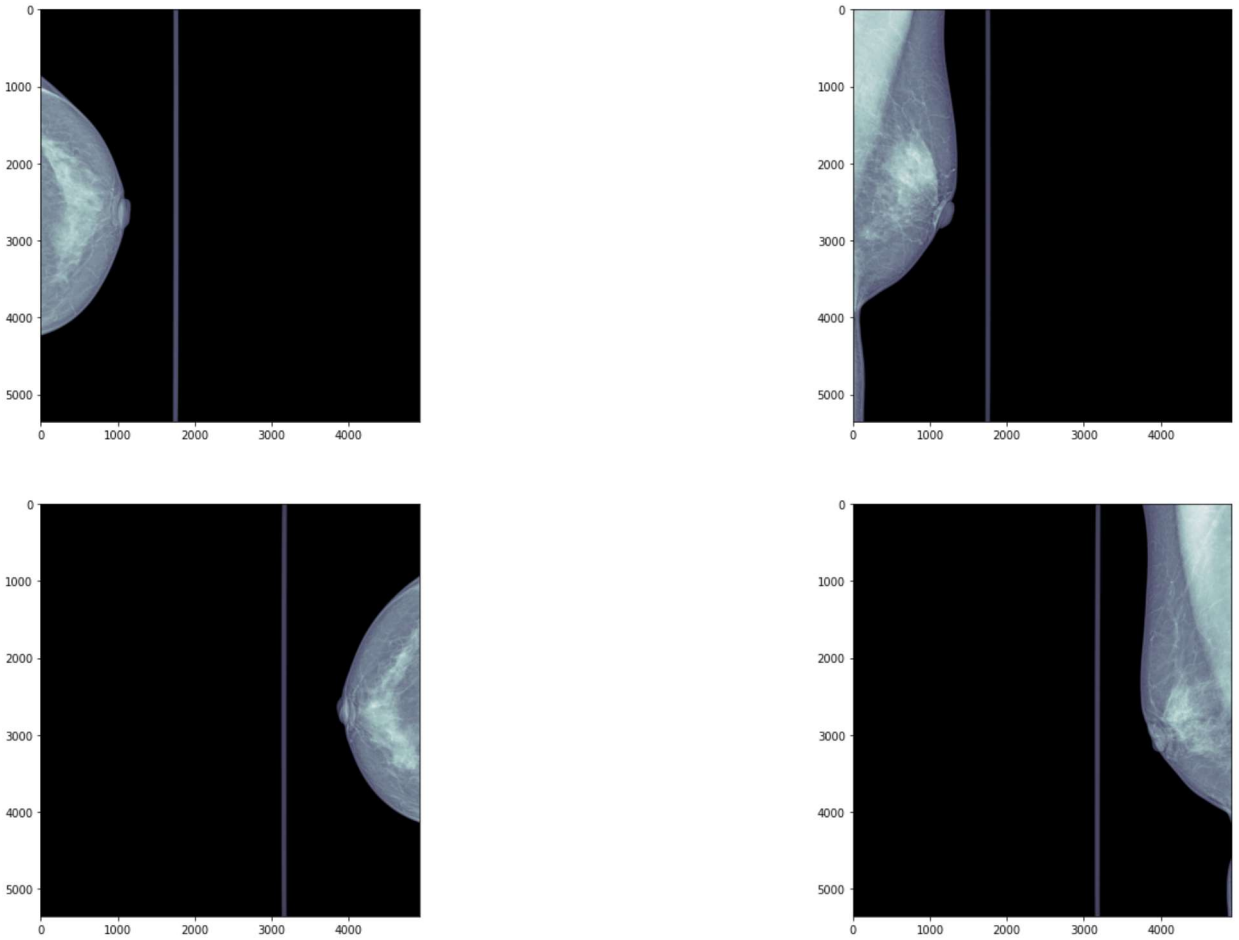
In [9]:
```python
def rescale_img_to_hu(dcm_ds):
    """Rescales the image to Hounsfield unit."""
    data = dcm_ds.pixel_array
    if dcm_ds.PhotometricInterpretation == "MONOCHROME1":
        data = np.amax(data) - data
    return data * dcm_ds.RescaleSlope + dcm_ds.RescaleIntercept
```

In [10]:
```python
def show_images_for_patient(patient_id):
    patient_dir = os.path.join('../input/rsna-breast-cancer-detection/train_images', s
    num_images = len(glob.glob(f"{patient_dir}/*"))
    print(f"Number of images for patient: {num_images}")
    fig, axs = plt.subplots(2, 2, figsize=(24,15))
    axs = axs.flatten()
    for i, img_path in enumerate(list(Path(patient_dir).iterdir())):
```

```
        ds = pydicom.dcmread(img_path)
        axs[i].imshow(rescale_img_to_hu(ds), cmap="bone")
```

In [11]:
```
show_images_for_patient(10006)
```

```
Number of images for patient: 4
```



From the plots, it's evident that the images are of considerable size. Specifically, the breast occupies only a small portion of the image. Therefore, especially considering the limited compute budget, it's highly advisable to explore methods for cropping out the image portions that do not contain pertinent information. The divider line can play an important role in this process.

In [12]:
```
train_csv = pd.read_csv('../input/rsna-breast-cancer-detection/train.csv')
train_csv.head()
```

Out[12]:

	site_id	patient_id	image_id	laterality	view	age	cancer	biopsy	invasive	BIRADS	implant
0	2	10006	462822612	L	CC	61.0	0	0	0	NaN	0
1	2	10006	1459541791	L	MLO	61.0	0	0	0	NaN	0
2	2	10006	1864590858	R	MLO	61.0	0	0	0	NaN	0
3	2	10006	1874946579	R	CC	61.0	0	0	0	NaN	0
4	2	10011	220375232	L	CC	55.0	0	0	0	0.0	0

The targets are stored in the train.csv file, along with additional metadata. Specifically, they are stored in the cancer column. Furthermore, the file provides a mapping of images to their laterality, indicating whether an image is of the left or right breast. This can be incredibly useful for training purposes.

As for information on laterality (and other metadata) in the test set, it is contained in the test.csv file.

In [13]:
```
test_csv = pd.read_csv('../input/rsna-breast-cancer-detection/test.csv')
test_csv
```

Out[13]:

	site_id	patient_id	image_id	laterality	view	age	implant	machine_id	prediction_id
0	2	10008	736471439	L	MLO	81	0	21	10008_L
1	2	10008	1591370361	L	CC	81	0	21	10008_L
2	2	10008	68070693	R	MLO	81	0	21	10008_R
3	2	10008	361203119	R	CC	81	0	21	10008_R

As anticipated, the test file does not include the cancer column. For training purposes, the number of images available can be determined from the train.csv file.

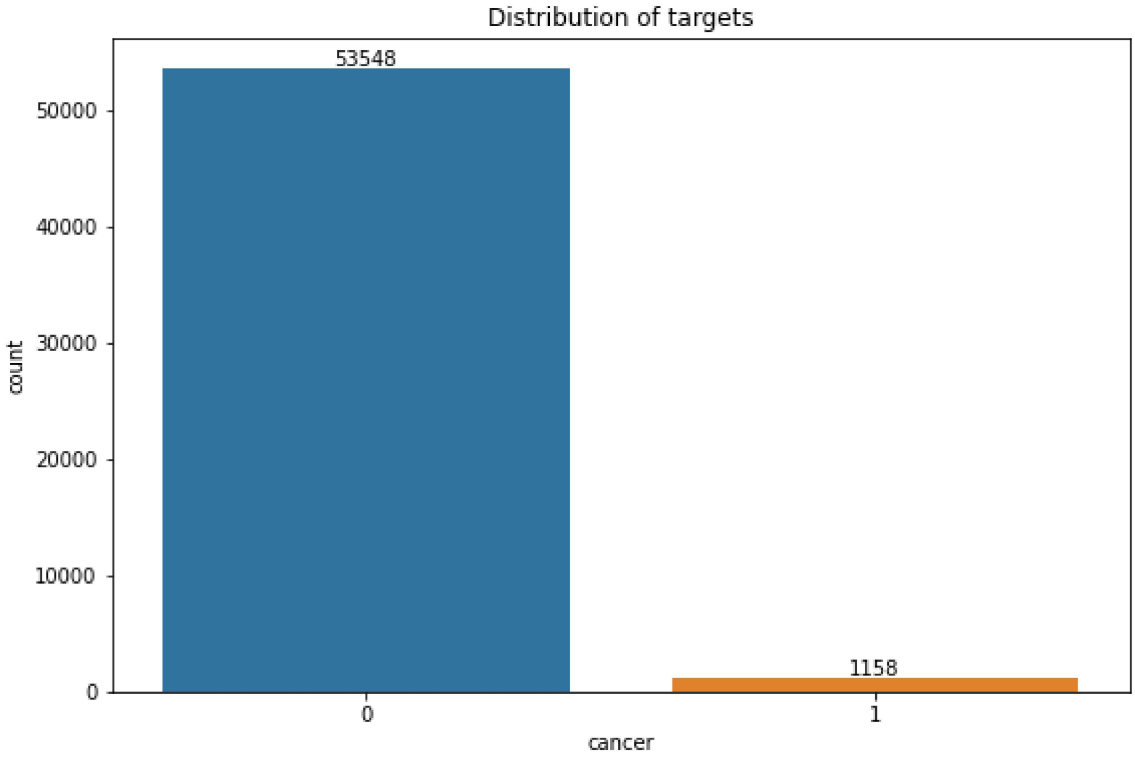
In [14]:
```
train_csv.shape[0], train_csv.patient_id.nunique()
```

Out[14]:
```
(54706, 11913)
```

There are a total of 54,706 images in the training set, spread across 11,913 patients.

Distribution of labels

In [15]:
```
plt.figure(figsize=(20,6))
plt.subplot(1,2,1)
ax1 = sns.countplot(data=train_csv, x='cancer')
for container in ax1.containers:
    ax1.bar_label(container)
plt.title('Distribution of targets');
```



The classes exhibit high imbalance. It's probable that we'll need to tackle this during training, possibly through methods such as upsampling the 'cancer' class, downsampling the 'cancer absent' class, or adjusting class weights.

Training a fast.ai model

Fast.ai offers a collection of pre-built models, cutting-edge techniques, and streamlined APIs that facilitate complex tasks in deep learning, such as image classification, natural language processing, tabular data analysis, and collaborative filtering.

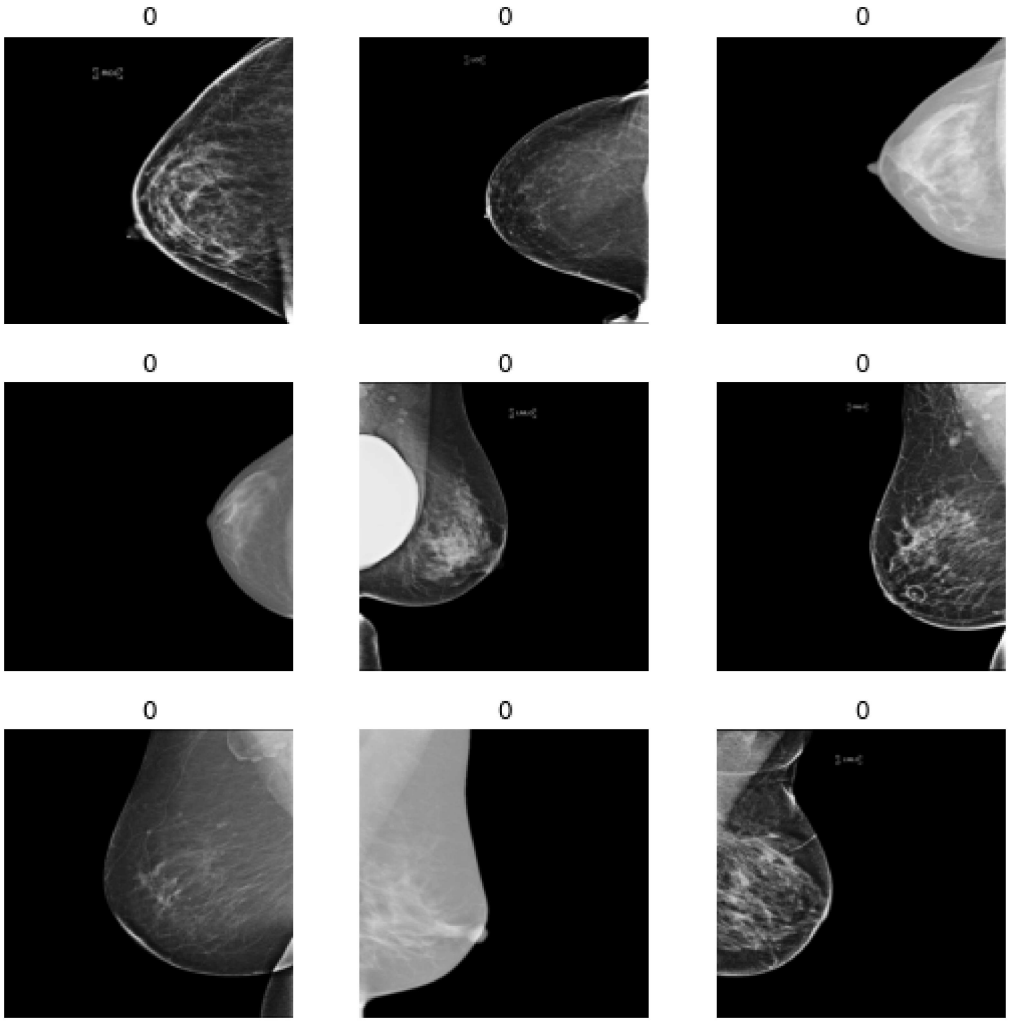
Creating dataloaders

In [16]:
```
from fastai.data.all import *
from fastai.vision.all import *

path = '../input/rsna-mammography-images-as-pngs/images_as_pngs/train_images_processed

train_csv = pd.read_csv('../input/rsna-breast-cancer-detection/train.csv')
fn2label = {fn: cancer_or_not for fn, cancer_or_not in zip(train_csv['image_id'].astyp

def label_func(path):
    return fn2label[path.stem]

dblock = DataBlock(
    blocks    = (ImageBlock, CategoryBlock),
    get_items = get_image_files,
    get_y = label_func,
    splitter  = RandomSplitter()
```

```
)
dsets = dblock.datasets(path)
dls = dblock.dataloaders(path)
```

And this is what a batch of our data will look like.

In [17]:
```
dls.show_batch()
```



Creating the learner and training for a single epoch

In [18]:
```
learn = vision_learner(dls, resnet18, metrics=error_rate, pretrained=False)
learn.fit_one_cycle(1, 1e-2)
```

epoch	train_loss	valid_loss	error_rate	time
0	0.112463	0.108502	0.021662	02:30

The result example

Let's explore the process of creating an example result to better understand the expected format of predictions (and how to generate them). This example will follow the structure of the stub test set we currently have in place, encompassing more than just a single directory.

Here is the current setup of the test set:

In [19]:
```
ls ../input/rsna-breast-cancer-detection/test_images
```

```
10008/
```

Let's set up all the necessary tools for generating predictions and verify their functionality by producing a random prediction composed of floats ranging between 0 and 1.

In [20]:
```
result = pd.DataFrame(data={'prediction_id': test_csv['prediction_id'], 'cancer': np.r
result.head()
```

Out[20]:

	prediction_id	cancer
0	10008_L	0.850131
2	10008_R	0.836814

In [21]:
```
result.to_csv('result.csv', index=False)
```