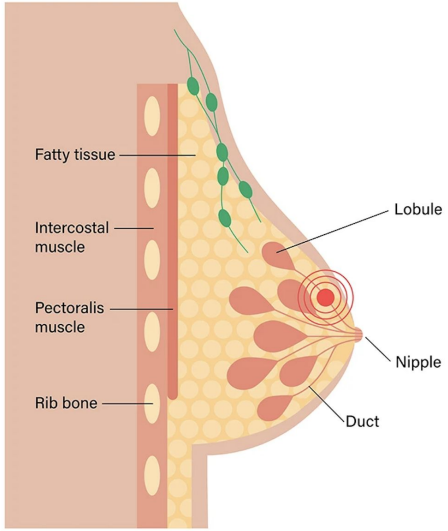
Breast Cancer Detection

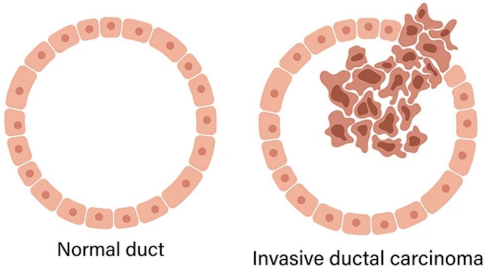
Aquincum Labs

Isaac Klar - Mingi Kang - Karam Al-Askar - Shaamil Karim

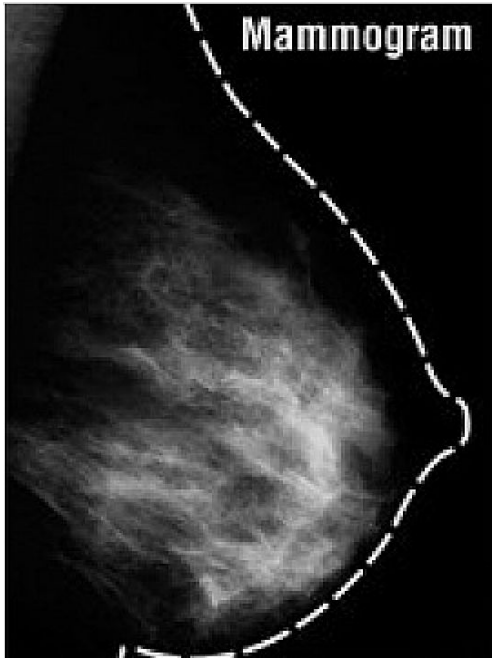
Introduction

What is Breast Cancer?





- Invasive ductal carcinoma (IDC)
- A type of cancer that starts in the milk ducts of the breast and moves into nearby tissue.
- IDC Positive = BAD
- IDC Negative = GOOD



Why detect Breast Cancer?

- IDC is the most common cancer among women worldwide
- Traditional screening methods like mammograms, lead to false positives or negatives
- Inaccess to modern screening methods in some countries
- With data science we can:
  - leverage large datasets and advanced algorithms
  - Develop tools to enhance detection accuracy
  - Make breast cancer detection more accessible

Next Up

- Data Processing & Analysis
- Initial Model
- Improved Model

Data Processing + Analysis

```python
import kagglehub

# Download latest version
path = kagglehub.dataset_download("paultimothymooney/breast-histopathology-images")

print("Path to dataset files:", path)
```

```
Download already complete (3326820824 bytes).
Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/paultimothymooney/breast-histo
```

```python
import os

dataPath = os.path.join(os.path.expanduser("~"), ".cache/kagglehub/datasets/paultimothymoor
print(dataPath)
```

```
/root/.cache/kagglehub/datasets/paultimothymooney/breast-histopathology-images/versions/1
```

Downloading data

- Download data from Kaggle
- Kaggle Hub Python Library

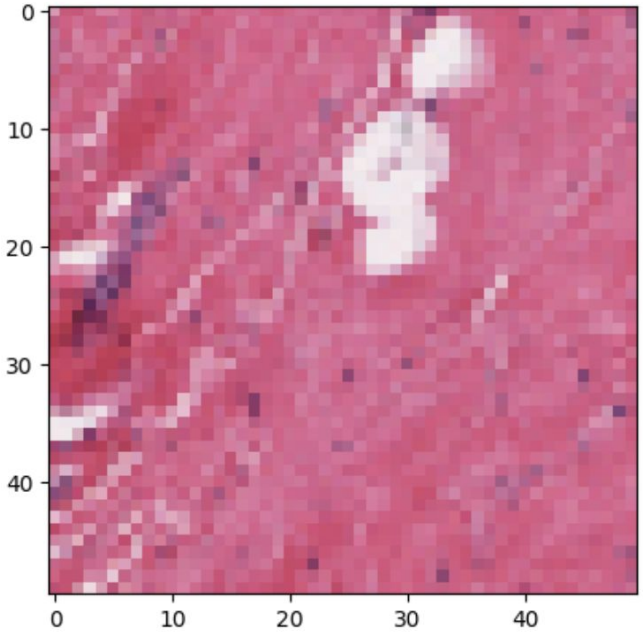
Data Handling

- Format: “u_xX_yYclassC.png”
- Number of IDC negative patches: 198,738
- Number of IDC positive patches: 78,786
- Total number of patches: 277,524

```python
# Get all the paths to the images
# -> glob allows for pattern matching to get all the photo paths
idc_neg = glob.glob(dataPath +'/*/0/*.png', recursive = True)
idc_pos = glob.glob(dataPath +'/*/1/*.png', recursive = True)

# Total : 277,524 patches of size 50 x 50 (198,738 IDC negative and 78,786 IDC positive)
print("Number of IDC negative patches: ", len(idc_neg))
print("Number of IDC positive patches: ", len(idc_pos))
print("Total number of patches: ", len(idc_neg) + len(idc_pos))
```

```
Number of IDC negative patches:  198738
Number of IDC positive patches:  78786
Total number of patches:  277524
```

```python
# Function to convert the image to a tensor
import cv2

def image_to_numpy(image_path, target_size=(50, 50)):
    # Load the image in BGR format
    img = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if img is None:
        raise ValueError(f"Image not found or unable to load: {image_path}")

    # Convert BGR to RGB
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Resize the image to the target size
    img_resized = cv2.resize(img_rgb, target_size, interpolation=cv2.INTER_AREA)

    return img_resized / 255

# Example usage
example_image_path = '/8863/0/8863_idx5_x51_y1251_class0.png'
example_tensor = image_to_numpy(dataPath + example_image_path)
print("Shape of the tensor: ", example_tensor.shape)
```



Data Visualization

- Cv2 library
- Color = cv2.COLOR_BGR@RGB
- matplotlib.pyplot

```
# Function to plot the image
def plot_image(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img_rgb)

# Example usage
plot_image(idc_neg[0])
```

```python
# Function to convert the image to a tensor
import cv2

def image_to_numpy(image_path, target_size=(50, 50)):
    # Load the image in BGR format
    img = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if img is None:
        raise ValueError(f"Image not found or unable to load: {image_path}")

    # Convert BGR to RGB
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Resize the image to the target size
    img_resized = cv2.resize(img_rgb, target_size, interpolation=cv2.INTER_AREA)

    return img_resized / 255

# Example usage
example_image_path = '/8863/0/8863_idx5_x51_y1251_class0.png'
example_tensor = image_to_numpy(dataPath + example_image_path)
print("Shape of the tensor: ", example_tensor.shape)
```

```
Shape of the tensor:  (50, 50, 3)
```

Data Conversion

- Initially believed that we needed to convert .png to tensors (too long + memory intensive)
- .png -> NumPy Array
- NumPy arrays are less computationally expensive

Data Preparation + Analysis for Models

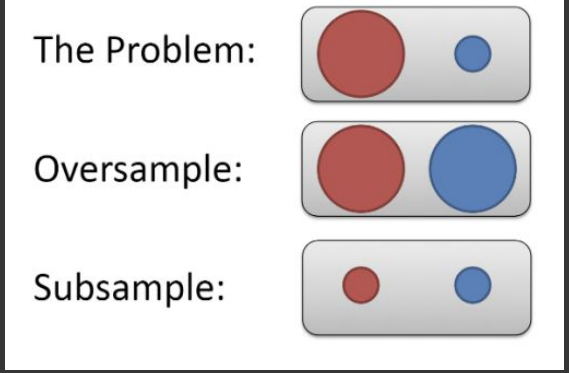
```python
# Get 12000 random idc negative image and convert to numpy arrays
idc_neg_numpy = []

for i in random.sample(range(0, len(idc_neg)), 12000):
    idc_neg_numpy.append(image_to_numpy(idc_neg[i]))

# Get 12000 random idc positive image and convert to numpy arrays
idc_pos_numpy = []

for i in random.sample(range(0, len(idc_pos)), 12000):
    idc_pos_numpy.append(image_to_numpy(idc_pos[i]))

# We combine the negative and positive data and create a y lable numpy array.
X = np.array(idc_neg_numpy + idc_pos_numpy)
y = np.array([0] * len(idc_neg_numpy) + [1] * len(idc_pos_numpy))
print(X.shape)
print(y.shape)
print()

# get X train, X test, y train, y test with sklearn.model_selection's train_test_split
# test split is 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

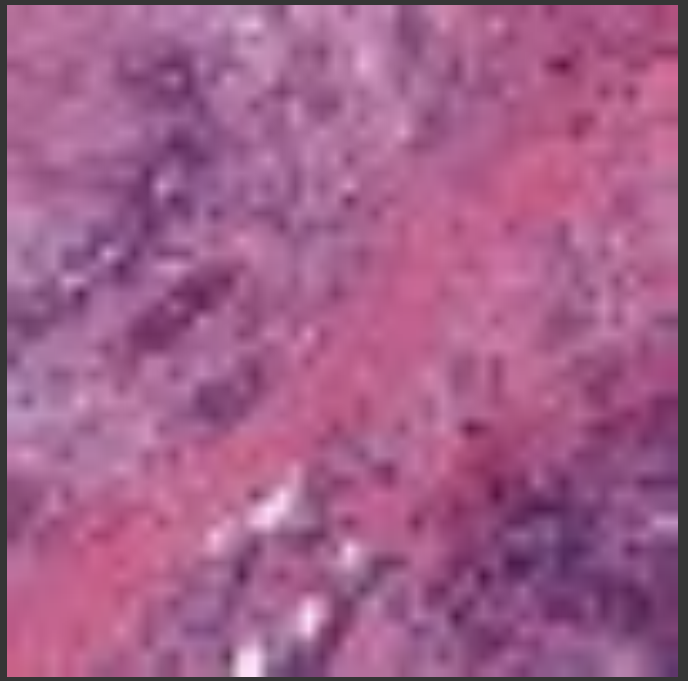
```
(24000, 50, 50, 3)
(24000,)

(19200, 50, 50, 3)
(19200,)
(4800, 50, 50, 3)
(4800,)
```

Imbalanced Data

Under-sampling / Sub-sampling

- 12,000 Data Points





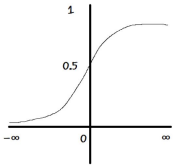
Initial Model

Initial Model

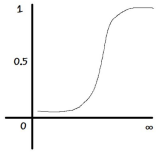
For our initial model, we used the same model that we used in class to classify digits, changing necessary attributes.

```
# In Class Model for Hand Written Digit Classification
model = Sequential()

model.add(Conv2D(32, (5, 5), strides=(1, 1), input_shape=(28, 28, 1), activation='relu',
data_format="channels_last"))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# Modified Model for Breast Cancer Classification
model1 = Sequential()

model1.add(Conv2D(32, (5, 5), strides=(1, 1), input_shape=(50, 50, 3), activation='relu',
data_format="channels_last"))
model1.add(Conv2D(64, (5, 5), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model1.add(Flatten())
model1.add(Dense(1, activation='sigmoid'))

model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```



Softmax



Sigmoid

Walkthrough model 1

```
#Initialize the Sequential model:
model1 = Sequential()

model1.add(Conv2D(32, (5, 5), strides=(1, 1), input_shape=(50, 50, 3), activation='relu', data_format="channels_last"))
model1.add(Conv2D(64, (5, 5), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model1.add(Flatten())
model1.add(Dense(1, activation='sigmoid'))

#Compiling the model, loss: categorical crossentropy, it is the most popular for these kind of problems,
#optimizer: adam, a faster variant of the stochastic gradient method
#metrics: accuracy (We want to know the accuracy after each epoch.)
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Results

Test accuracy: 0.794583

Precision 0.7946

Recall 0.794599

f1_score 0.7946

Confusion matrix:

1910 (TN)	481 (TP)
505 (FN)	1904 (FP)

Improved Model

Model 2

At an attempt to improve our model, we decided to implement the following changes:

- **Convolutional Blocks**
  Highlight what's new, unusual, or surprising.

- **Regularization**
  Added in dropout layers with rates between 0.2 and 0.5. We also added L2 regularization in the dense layer to penalize large weights.

- **2D Max Pooling**

  Extract the most important parts of each image.

- **Global Average Pooling**
  Uses global average pooling to aggregate the feature maps before the dense layer.

```python
model2 = Sequential()

# First Conv Block
model2.add(Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 3)))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))

# Second Conv Block
model2.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model2.add(Dropout(0.2))

# Third Conv Block
model2.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))

model2.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model2.add(Dropout(0.2))

model2.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))
```

```python
model2.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model2.add(Dropout(0.2))

model2.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))

model2.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.2))

# Global Average Pooling + Fully Connected
model2.add(GlobalAveragePooling2D())
model2.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model2.add(Dropout(0.5))
model2.add(Dense(1, activation='sigmoid'))

# Compile the model
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', 'Precision', 'Recall'])
```

Conclusion