LAB 5 : Which CNN Model is Appropriate to Diagnose or Classify Breast Cancer?

Date: 2024-June-24

Class: 2024-1 Image Processing with Deep Learning by Y.K Kim

Author: Eunji Ko 22100034, Garam Jin 21900727

Github:

GitHub - JinGaram/DLIP

Contribute to JinGaram/DLIP development by creating an account on GitHub.

https://github.com/JinGaram/DLIP/tree/master

JinGaram/**DLIP**

1 Contributor 0 Issues 0 Stars 0 Forks

DLIP/Final Project at main · eunjijuliako/DLIP

Contribute to eunjijuliako/DLIP development by creating an account on GitHub.

https://github.com/eunjijuliako/DLIP/tree/main/Final%20Project

eunjijuliako/**DLIP**

1 Contributor 0 Issues 0 Stars 0 Forks

Demo Video:

https://www.youtube.com/watch?v=B5pAhuRVq4M

Introduction

Background and Problem Statement

This final project aims to implement a purposeful algorithm with images using deep learning and image processing. Through this project, we wanted to know how to use image processing as a medical technology for medical diagnosis and analysis. Therefore, we focused on breast cancer problems and deep-learning CNN models.

First, breast cancer in Korea is the most common cancer among women, and the number of breast cancer patients has more than doubled over the past decade. It is among the highest incidence groups among Asian countries. Therefore, the importance of diagnosing and treating breast cancer is increasing. Second, there are many kinds of CNN models, but studies on which models are suitable for medical imaging are needed.

CNN Models

The CNN model, short for Convolutional Neural Network, is used when there is a large amount of complex data such as image data because operations are repeated for hundreds of layers. The CNN model can visually detect the presence or absence of cancer cells in images by reviewing thousands of data, or can be used in audio processing, object detection, synthetic data generation, and various fields. To this end, a pre-trained model or a model built by learning and testing a new model is used. We used three kind of CNN models in this project.

Yolo

: You Look Only Once

YOLOv8 provides the latest performance in terms of accuracy and speed when detecting and segmenting objects. With one stage detection method, YOLO can detect objects in real time by simultaneously performing classification and localization. There are few background errors because it learns the surrounding information and processes the entire image. As its name implies, the entire image is viewed only once. This means that it does not do the work of analyzing multiple images by dividing them into multiple sheets like the R-CNN-based method.

Architecture

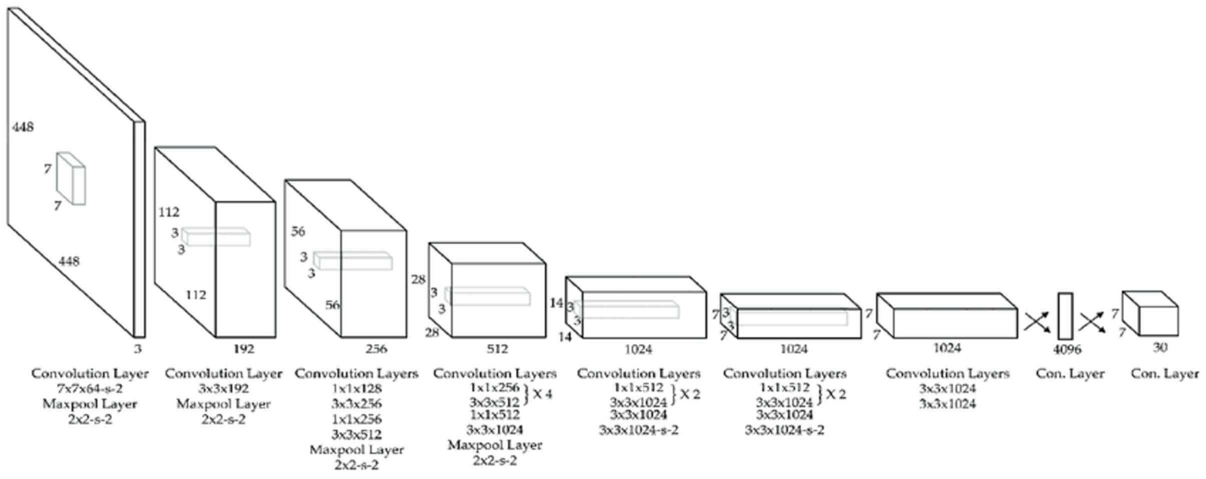


Figure 1. Yolo architecture

- Resizes the input image into 448×448 before going through the convolutional network.
- A 1×1 convolution is first applied to reduce the number of channels, which is then followed by a 3×3 convolution to generate a cuboidal output.
- The activation function under the hood is ReLU, except for the final layer, which uses a linear activation function.
- Some additional techniques, such as batch normalization and dropout, respectively regularize the model and prevent it from overfitting.

UNet

U-Net is a structure proposed in the paper 'U-Net: Convolutional Networks for Biomedical Image Segmentation', showing accurate image segmentation performance with very few training data, and won the ISBI Cell Tracking Challenge when 2015 by a large margin.

Architecture

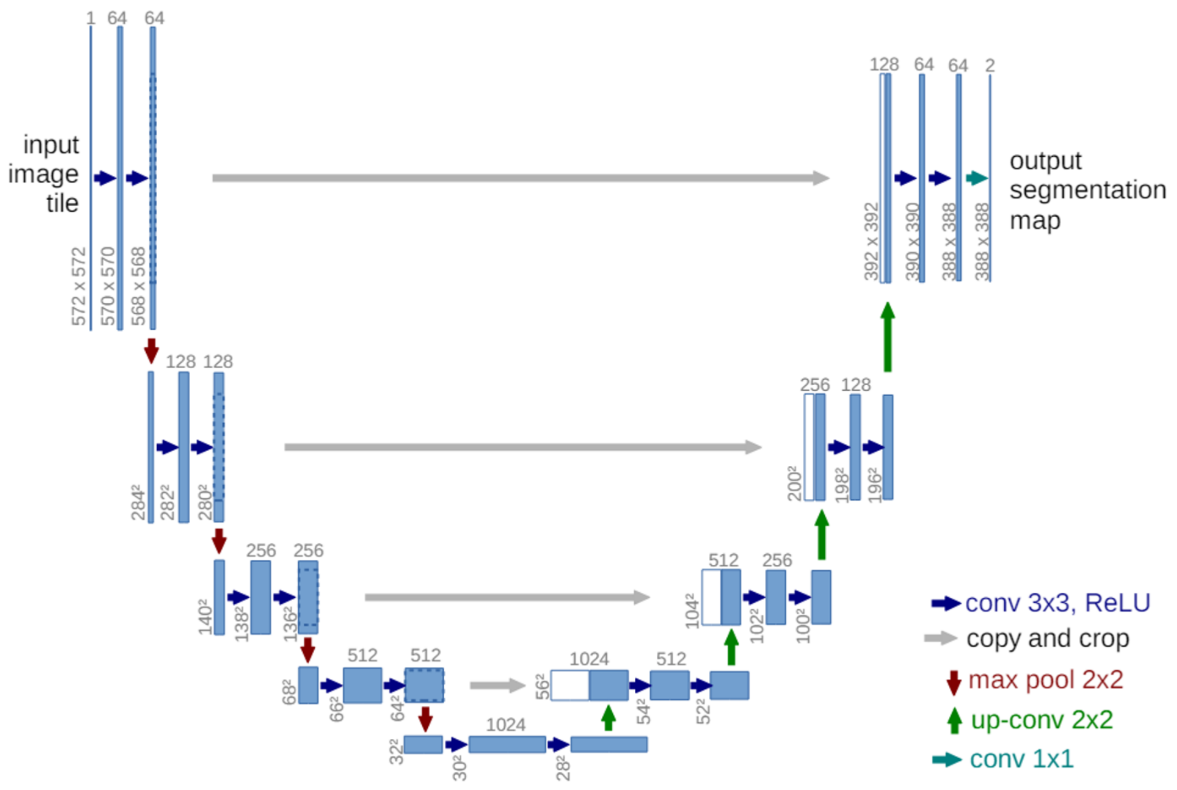


Figure 2. UNet architecture

- Overlap-tile strategy & Mirroring Extrapolation

It is a better method than zero padding for medical data such as cells that do not significantly affect left and right symmetries because it can give the effect of data augmentation while mirror extrapolation.

- Data Augmentation

  Elastic Deformation is a way of transforming pixel to randomly twist in different directions. If you look at the figure above, you can see that even elastic deformation transforms to be as real as it is in the real world. The authors used these techniques to increase data in the article.

FCN

: Fully Connected Networks for Semantic Segmentation

FCN is a model modified to perform semantic segmentation tasks using existing networks AlexNet, VGGNet, and GoogleNet whose performance has been verified in segmentation. In the existing network structure, a fully connected layer is inserted at the end, and in order to use this FCL, only a fixed size of input must be received, and after passing FCL, location information disappears. It is not suitable for image segmentation because segmentation requires location information (where an object exists). To overcome the disadvantages, FCN was created to replace the last FCL of the model used in segmentation with a convolution layer. When FCN is used, location information or class information is not lost and input size is not limited.

Architecture

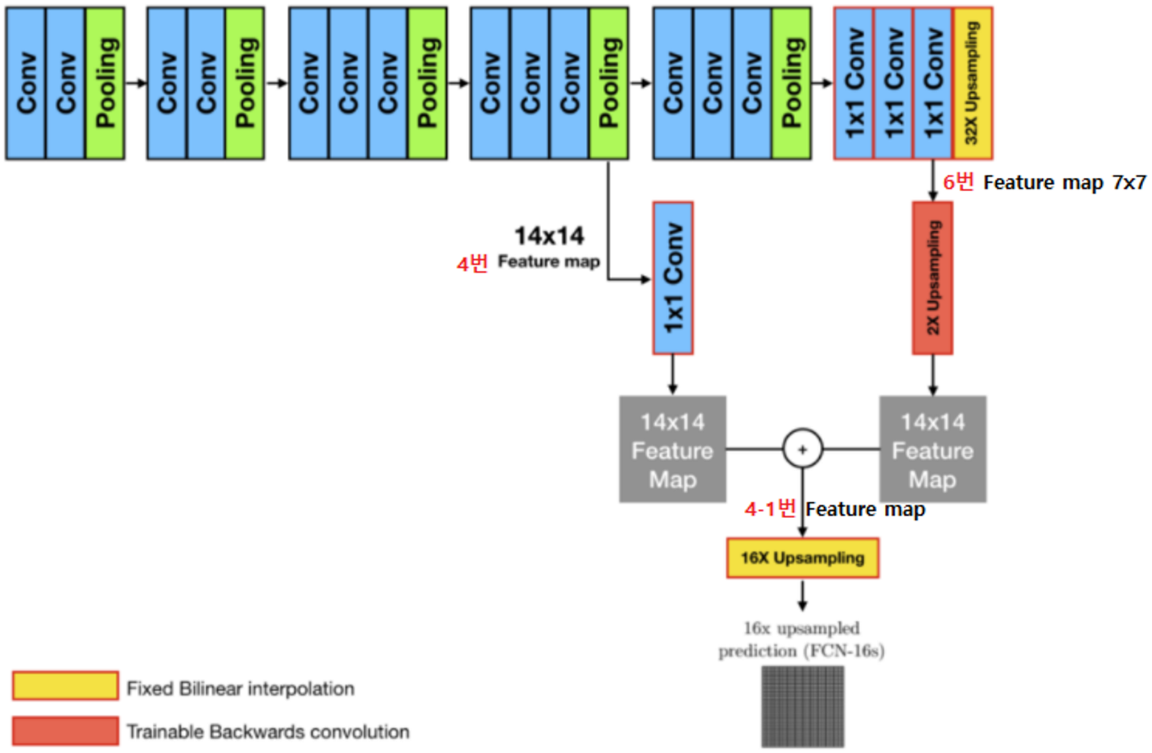


Figure 3. FCN architecture

1. Extracting Features via the Convolution Layer
2. Change the number of channels in the Feature Map to equal the number of clades in the dataset using the 1×1 convolution layer
3. Upsampling a low-resolution Heat Map using Transposed Convolution, and creating a Map of the same size as the input image
4. Color according to each Pixel class in the Map and return the result

Objective

In this project, we will find which CNN model is appropriate to diagnose or classify breast cancer through comparing the precision, recall, and accuracy for each models, which is yolo, UNet, and FCN.

Preparation

Software Installation

- OpenCV 4.90
- Visual Studio Code 2022
- GPU

- Pytorch

Dataset

- **Dataset link:** https://www.kaggle.com/datasets/vuppalaadithyasairam/ultrasound-breast-images-for-breast-cancer

Algorithm

Flowchart

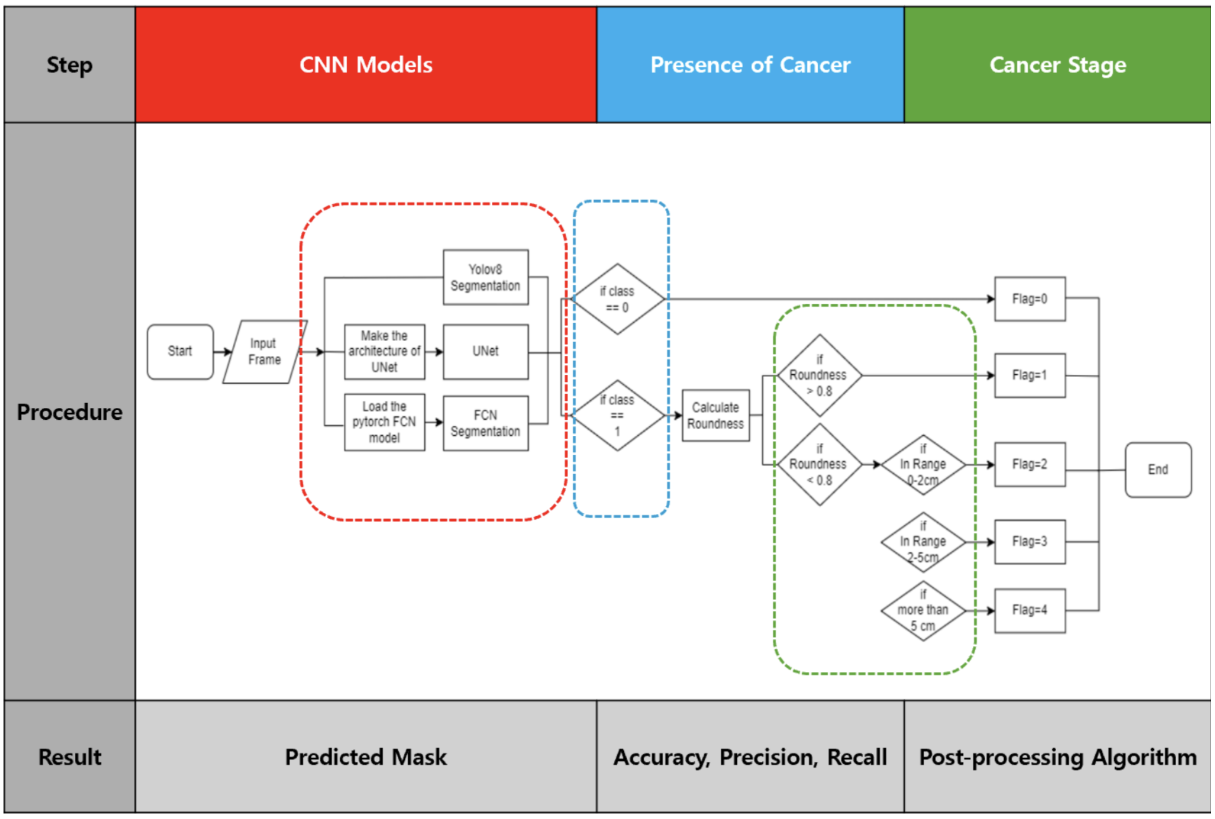


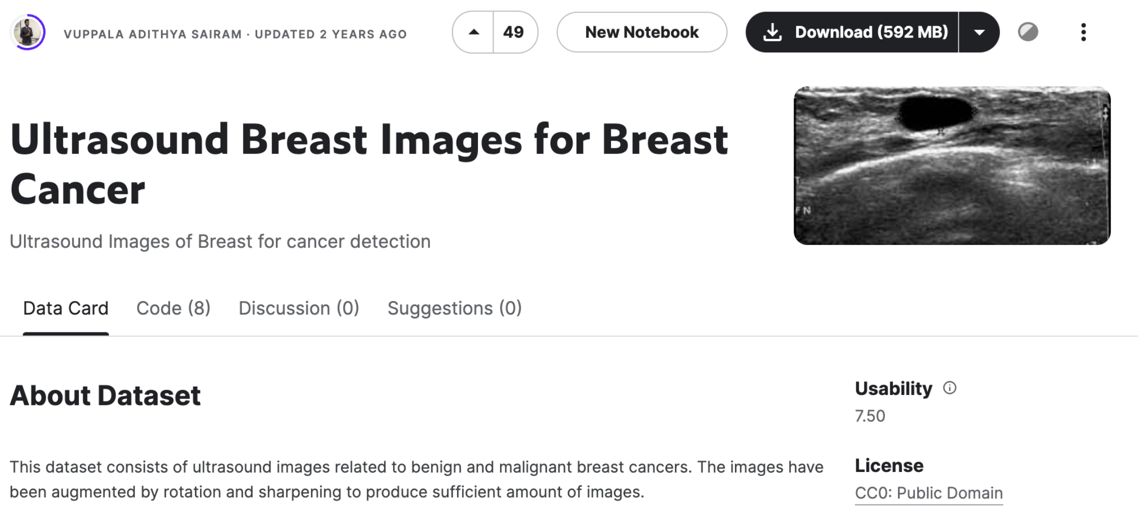
Figure 4. Flow chart

1. CNN models: Train the three CNN models with same dataset
2. Presence of Cancer: Depending on the class, there is no cancer when class is 0, and there is cancer when class is 1.
3. Cancer Stage: Depending on the roundness and the size of range, we can determine the stage of cancer. This is post-processing.

Procedure

1. Prepare the Data set

a. **Download the Images from Dataset link**



Ultrasound Breast Images for Breast Cancer

Ultrasound Images of Breast for cancer detection

Data Card Code (8) Discussion (0) Suggestions (0)

About Dataset

This dataset consists of ultrasound images related to benign and malignant breast cancers. The images have been augmented by rotation and sharpening to produce sufficient amount of images.

Usability
7.50

License
CC0: Public Domain

Figure 5. Data set

b. **Create or load the model**

- Yolo

```
from ultralytics import YOLO
model = YOLO('yolov8n-seg.pt')
```

We used the segmentation version 8n provided by ultralytics for the Yolo model.

- UNet

```
def unet_model(input_size=(128, 128, 3)):
    inputs = Input(input_size)

    # Contracting path
    conv1 = Conv2D(64, 3, activation='relu', padding='same
    conv1 = Conv2D(64, 3, activation='relu', padding='same
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='sam
    conv2 = Conv2D(128, 3, activation='relu', padding='sam
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same
    conv3 = Conv2D(256, 3, activation='relu', padding='same
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation='relu', padding='same
    conv4 = Conv2D(512, 3, activation='relu', padding='same
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = Conv2D(1024, 3, activation='relu', padding='sa
    conv5 = Conv2D(1024, 3, activation='relu', padding='sa

    # Expansive path
    up6 = concatenate([UpSampling2D(size=(2, 2))(conv5), c
    conv6 = Conv2D(512, 3, activation='relu', padding='same
    conv6 = Conv2D(512, 3, activation='relu', padding='same

    up7 = concatenate([UpSampling2D(size=(2, 2))(conv6), c
    conv7 = Conv2D(256, 3, activation='relu', padding='same
    conv7 = Conv2D(256, 3, activation='relu', padding='same

    up8 = concatenate([UpSampling2D(size=(2, 2))(conv7), c
    conv8 = Conv2D(128, 3, activation='relu', padding='same
    conv8 = Conv2D(128, 3, activation='relu', padding='same

    up9 = concatenate([UpSampling2D(size=(2, 2))(conv8), c
    conv9 = Conv2D(64, 3, activation='relu', padding='same
    conv9 = Conv2D(64, 3, activation='relu', padding='same

    conv10 = Conv2D(1, 1, activation='sigmoid')(conv9)

    model = Model(inputs=[inputs], outputs=[conv10])

    model.compile(optimizer='adam', loss='binary_crossentr

    return model

model = unet_model(input_size=(image_size[0], image_size[
```

The UNet model was built by directly writing the architecture.

- FCN

```
import torch
import torchvision.transforms as transforms

device = torch.device('cuda' if torch.cuda.is_available()
model = models.segmentation.fcn_resnet50(pretrained=True)
model.classifier[4] = torch.nn.Conv2d(512, 3, kernel_size=
model = model.to(device)
```

For the FCN model, we used the segmentation model provided by Pytorch. We changed the depth of convolution 1 to 3.

c. **Prepare the dataset depends on the CNN model**

- Yolo

```
train: 'Datasets/images/training'
val: 'Datasets/images/validation'
train: 'Datasets/masks/training'
val: 'Datasets/masks/validation'
```

- UNet

```
image_path = 'Datasets/images'
mask_path = 'Datasets/masks'
```

- FCN

```
image_path = 'Datasets/images'
mask_path = 'Datasets/masks'
```

The data was split into training and validation sets with a ratio of 8:2. Then, Yolo was set up with the following folder configuration. UNet was trained using the entire image. Similarly, FCN also used the entire image for training. Test images were selected randomly, using 20% of the data.

2. Training

a. **Train with the dataset**

- Yolo

```
from ultralytics import YOLO

def train():
    # Load a pretrained YOLO model
    model = YOLO('yolov8n-seg.pt')

    # Train the model using the 'maskdataset.yaml' dataset fo
    results = model.train(data='Yolo_segmentation.yaml', epoc

if __name__ == '__main__':
    train()
```

- UNet

```
# Path
image_path = 'Datasets/images'
mask_path = 'Datasets/masks'
image_size = (128, 128)

# Load Images
X, y = load_images(image_path, mask_path, image_size)

# Create model
model = unet_model(input_size=(image_size[0], image_size[1], 

# Callback
checkpoint = ModelCheckpoint('unet.keras', monitor='val_loss'
early_stopping = EarlyStopping(monitor='val_loss', patience=1

# Train
history = model.fit(X, y, validation_split=0.1, epochs=50, ba
# Save
model.save('unet_final.keras')
```

- FCN

```
# Training Function
def train_model(model, dataloader, criterion, optimizer, num_e
    model.train()
    for epoch in range(num_epochs):
        running_loss = 0.0
        for images, labels in dataloader:
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(images)['out']
            labels = labels.float()
            loss = criterion(outputs, labels)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * images.size(0)

        epoch_loss = running_loss / len(dataloader.dataset)
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {epoch_los

# Training the Model
num_epochs = 12
train_model(model, train_dataloader, criterion, optimizer, nun
print("Done!")
```

The models mentioned above were trained using 12 epochs in common. For Yolo, the trained model was saved as a '.pt' file. The UNet model was saved as a '.keras' file. Finally, the FCN model was saved as a '.pth' file.

3. Post-processing

```
def determine_class(mask):
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.

    if len(contours) == 0:
        return 1, 0  # normal (no mask detected)

    contour = contours[0]

    perimeter = cv2.arcLength(contour, True)
    area = cv2.contourArea(contour)
    if perimeter == 0:
        return 1, 0  # normal (no valid contour)

    circularity = 4 * np.pi * (area / (perimeter * perimeter))
    if circularity > 0.8:
        return 2, area  # benign (round mask)

    else:
        return 3, area  # malignant (irregular mask)

# Class info
class_info = {
    1: {'color': (0, 1, 0), 'name': 'normal'},      # normal (gr
    2: {'color': (1, 1, 0), 'name': 'benign'},      # benign (ye
    3: {'color': (1, 0, 0), 'name': 'malignant'}    # malignant
}

def visualize_results(images, model):
    fig, axs = plt.subplots(3, 3, figsize=(12, 12))

    for i, (image, _, img_name) in enumerate(images):
        row, col = divmod(i, 3)
        # Original image
        original_image = TF.to_pil_image(image)

        # Prediction
        with torch.no_grad():
            model_output = model(image.unsqueeze(0).to(device))[
            predicted_mask = torch.sigmoid(model_output)[0][0]  
            predicted_mask = (predicted_mask > 0.5).cpu().numpy(

        # Determine class and area
        class_id, area = determine_class(predicted_mask)
        color = class_info[class_id]['color']
        class_name = class_info[class_id]['name']

        # Determine stage for malignant class
        if class_id == 3:
            if area < 100:
                stage = '1'
            elif 100 <= area < 200:
                stage = '2'
            elif 200 <= area < 300:
                stage = '3'
            else:
                stage = '4'
            title = f'Class: {class_name}, Stage: {stage}'
        else:
            title = f'Class: {class_name}'
```

Post-processing was performed to utilize the trained models effectively. The classes were divided into three categories: normal, benign, and malignant.

Colors were assigned to distinguish each class. Additionally, the classes were differentiated based on the shape of the detected mask. If there was no mask, it was classified as normal.

If the mask was close to a circle, it was classified as benign. If the mask had a jagged shape, it was classified as malignant.

Further post-processing was performed for those classified as malignant. The stages were divided based on size, ranging from 1 to 4 from smallest to largest. This allows for the diagnosis of cancer and classification of cancer types and progression stages based on characteristics. The Yolo model was used without any post-processing.

4. Test & Evaluation

1. **Binary classification Evaluation**

```
# Load the trained model
model_path = ""
model = torch.load(model_path)
model.eval()

# Convert model to appropriate device
device = torch.device('cuda' if torch.cuda.is_available() else '
model = model.to(device)

# Function to calculate metrics for the entire dataset
def calculate_metrics_and_display(dataloader, model):
    pred_has_white_all = []
    label_has_white_all = []

    with torch.no_grad():
        for i, (images, labels) in enumerate(dataloader):
            images = images.to(device)
            outputs = model(images)['out']
            preds = torch.sigmoid(outputs) > 0.5

            for j in range(images.size(0)):
                pred = preds[j].cpu().numpy().astype(np.uint8)
                label = labels[j].cpu().numpy().astype(np.uint8)

                # Check for presence of white pixels (255)
                pred_has_white = np.sum(pred) > 0
                label_has_white = np.sum(label) > 0

                pred_has_white_all.append(pred_has_white)
                label_has_white_all.append(label_has_white)

                # Determine if the prediction is correct based o
                correct = pred_has_white == label_has_white
                result = "맞음" if correct else "틀림"

                # Print the results
                img_name = test_dataset.image_files[i * images.s
                print(f'파일명: {img_name}, 예측한 값: {pred_has_wh

    return np.array(pred_has_white_all), np.array(label_has_whit
# Calculate metrics for the test dataset and display results
all_preds, all_labels = calculate_metrics_and_display(test_datal

# Calculate Precision, Recall, and Accuracy
precision = precision_score(all_labels, all_preds, pos_label=Tru
recall = recall_score(all_labels, all_preds, pos_label=True)
accuracy = accuracy_score(all_labels, all_preds)

print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'Accuracy: {accuracy:.2f}')
```

2. **Post-processign Algorithm Evaluation**

```
def determine_class(mask):
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.

    if len(contours) == 0:
        return 1, 0  # normal (no mask detected)

    contour = contours[0]

    perimeter = cv2.arcLength(contour, True)
    area = cv2.contourArea(contour)
    if perimeter == 0:
        return 1, 0  # normal (no valid contour)

    circularity = 4 * np.pi * (area / (perimeter * perimeter))
    if circularity > 0.8:
        return 2, area  # benign (round mask)

    else:
        return 3, area  # malignant (irregular mask)

# Class info
class_info = {
    1: {'color': (0, 1, 0), 'name': 'normal'},      # normal (gr
    2: {'color': (1, 1, 0), 'name': 'benign'},    # benign (yell
    3: {'color': (1, 0, 0), 'name': 'malignant'}    # malignant
}

def parse_actual_class_from_filename(filename):
    if "normal" in filename:
        return 1
    elif "benign" in filename:
        return 2
    elif "malignant" in filename:
        return 3
    else:
        return None

def evaluate_results(images, model):
    y_true = []
    y_pred = []

    for image, _, img_name in images:
        # Prediction
        with torch.no_grad():
            model_output = model(image.unsqueeze(0).to(device))[
            predicted_mask = torch.sigmoid(model_output)[0][0]
            predicted_mask = (predicted_mask > 0.5).cpu().numpy(

        # Determine class and area
        class_id, _ = determine_class(predicted_mask)

        # Get actual class from filename
        actual_class = parse_actual_class_from_filename(img_name
        y_true.append(actual_class)
        y_pred.append(class_id)

        # Determine correctness
        correctness = "맞음" if actual_class == class_id else "틀
        print(f'{img_name}: {correctness}')

    # Calculate precision, recall, and accuracy
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    accuracy = accuracy_score(y_true, y_pred)
    print(f'(Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Number of Images: {len(images)}')

# Convert model to appropriate device
device = torch.device('cuda' if torch.cuda.is_available() else '
model = model.to(device)

# Evaluate results
evaluate_results(images_to_show, model)
```

3. **Segmentation Evaluation**

```
# Calculate & Output with result
def calculate_metrics(test_dataloader, model, device):
    gt_sizes = []
    pred_sizes = []
    results = []
    total = 0

    with torch.no_grad():
        for images, labels, img_name in test_dataloader:
            images = images.to(device)
            labels = labels.to(device) if labels is not None els

            outputs = model(images)['out']
            predicted_masks = torch.sigmoid(outputs)
            predicted_masks = (predicted_masks > 0.7).float()

            for label, predicted_mask in zip(labels, predicted_ma
                if label is not None:
                    gt_size = label.sum().item()
                else:
                    gt_size = 0
                pred_size = predicted_mask.sum().item()

                if gt_size == 0 and pred_size == 0:
                    results.append((img_name[0], pred_size, gt_s
                    continue

                total += 1
                if gt_size * 0.7 <= pred_size <= gt_size * 1.3:
                    results.append((img_name[0], pred_size, gt_si
                    gt_sizes.append(1)
                    pred_sizes.append(1)
                else:
                    results.append((img_name[0], pred_size, gt_si
                    gt_sizes.append(1)
                    pred_sizes.append(0)

    precision = precision_score(gt_sizes, pred_sizes)
    recall = recall_score(gt_sizes, pred_sizes)
    accuracy = accuracy_score(gt_sizes, pred_sizes)

    return precision, recall, accuracy, results, total

# Calculate
precision, recall, accuracy, results, total = calculate_metrics(

for img_name, pred_size, gt_size, result in results:
    print(f"File name: {img_name}, Predict: {pred_size}, Real: {

print(f"Total images compared: {total}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Accuracy: {accuracy:.4f}")
```

Three evaluation methods were employed accordingly. The first evaluation was binary, determining whether cancer was detected or not. The initial goal was to detect the presence of cancer. This was assessed by determining if the algorithm correctly identified cancer in images where it was present.

The second evaluation was conducted after post-processing. Since there is no information regarding the stage in the images, this could not be evaluated. Instead, the evaluation was based on classifying the detected cancer into three categories when detected.

The final evaluation was for segmentation. This was done because we used segmentation models. A good evaluation was given if the similarity to the actual mask size was 70% or more. Segmentation evaluation was deemed the most reliable among the three evaluation methods.

Analysis

Result & Discussion

1. **Binary classification Evaluation - Presence of cancer**

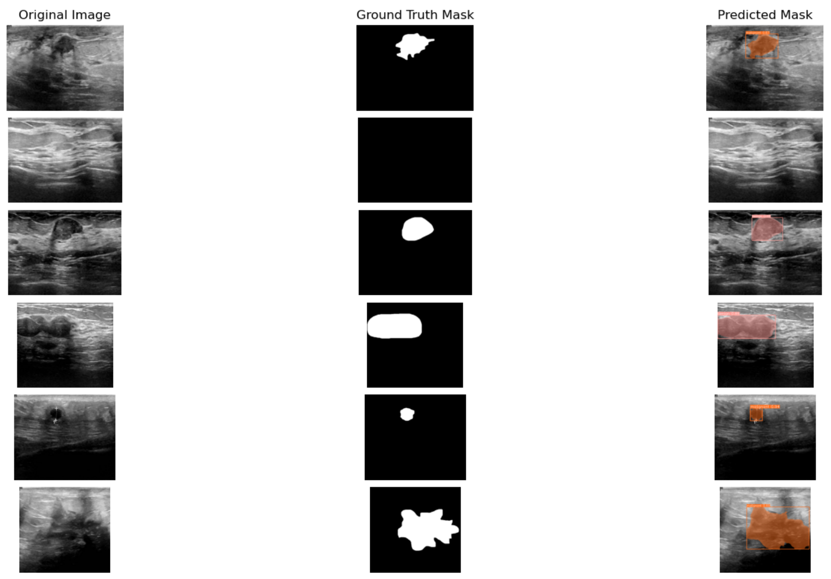


Figure 6. Yolo predicted mask result

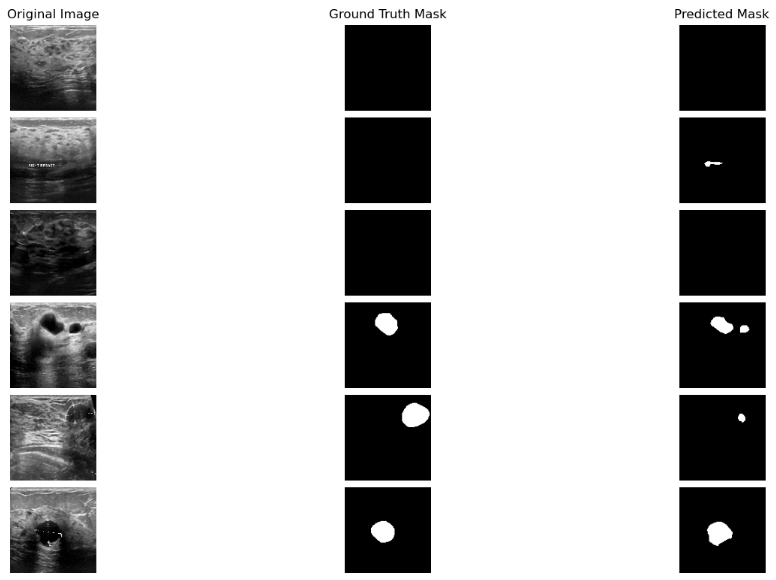


Figure 7. UNet predicted mask result

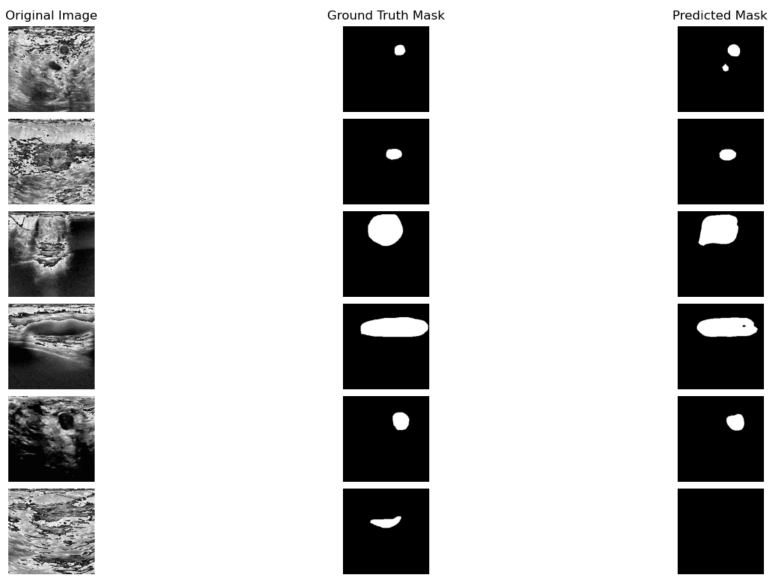


Figure 8. FCN predicted mask result

	Yolov8	UNet	FCN
Accuracy	100%	70%	88%
Precision	100%	92%	93%
Recall	100%	67%	91%

Overall, yolov8 detects well the presence of cancer. FCN performs better at detecting cancer than UNet in the perspective of accuracy, precision, and recall.

YOLO can outperform UNet or FCN in breast cancer ultrasound imaging because it can capture both regional and global features well with a structure specialized for object detection and an efficient learning method. These features can also be effectively applied in classification tasks.

For classification tasks, a classification-optimized model may be better suited than a segmentation model. Since UNet was originally developed for segmentation, it is possible that FCN specialized for classification tasks performed better. In addition, UNet usually has more parameters and complexity, and can cause overfitting problems. If the dataset is not large enough or diverse enough, UNet may overfit and degrade performance in test data.

2. **Post-processing Algorithm Evaluation - Stage of cancer**

   This could not be evaluated because there is no information about the cancer stage in the images.

3. **Segmentation Evaluation - Area of segmentation**

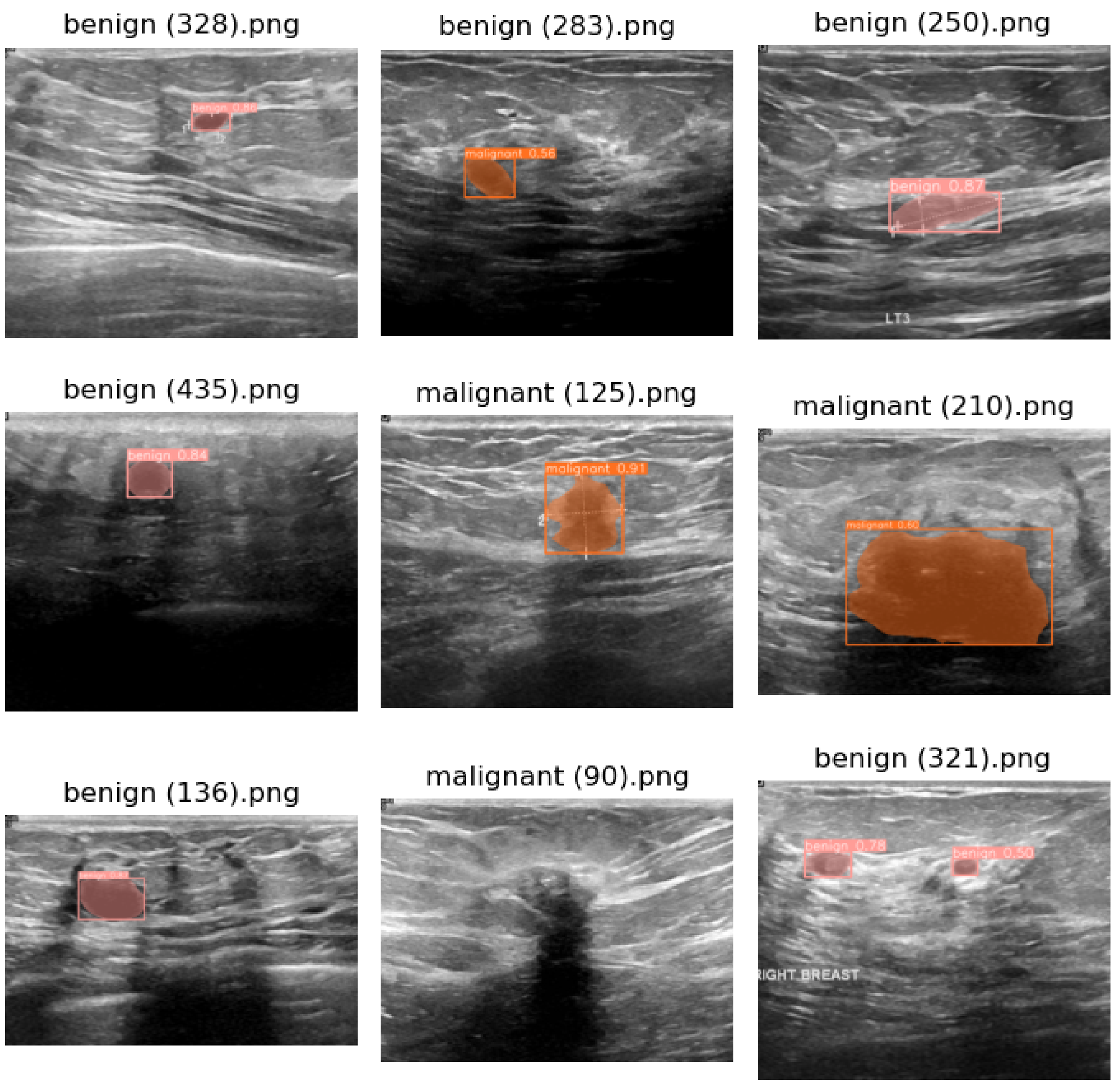


Figure 9. Yolo segmentation result

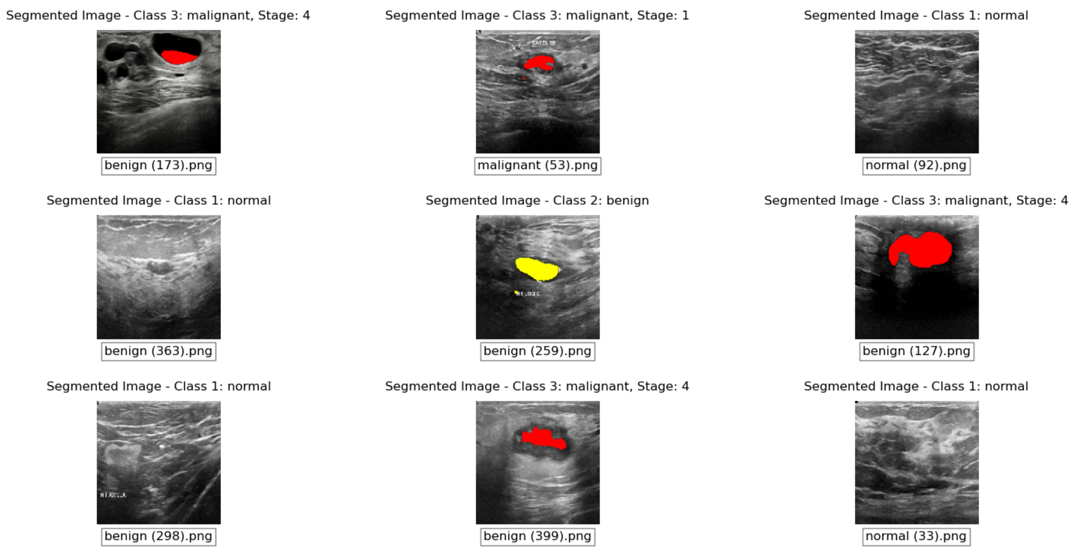

Segmented Image - Class 3: malignant, Stage: 4
benign (173).png
Segmented Image - Class 3: malignant, Stage: 1
malignant (53).png
Segmented Image - Class 1: normal
normal (92).png
Segmented Image - Class 1: normal
benign (363).png
Segmented Image - Class 2: benign
benign (259).png
Segmented Image - Class 3: malignant, Stage: 4
benign (127).png
Segmented Image - Class 1: normal
benign (298).png
Segmented Image - Class 3: malignant, Stage: 4
benign (399).png
Segmented Image - Class 1: normal
normal (33).png

Figure 10. UNet segmentation result

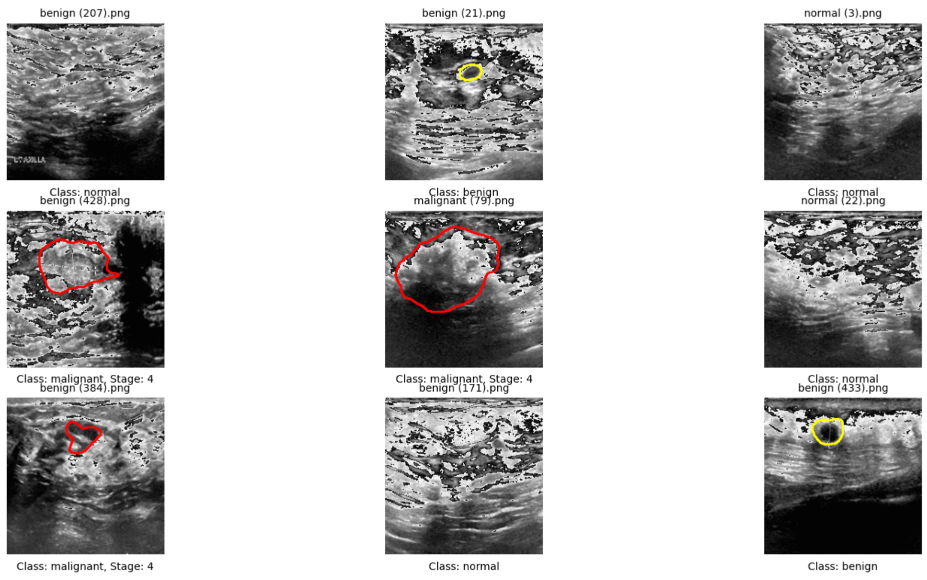


Figure 11. FCN segmentation result

	Yolov8	UNet	FCN
Accuracy	61%	15%	52%
Precision	100%	40%	100%
Recall	61%	14%	52%

YOLO showed the best performance in breast cancer ultrasound image segmentation because its model structure can perform segmentation by effectively combining the context and regional information of the entire image. FCN also performed well with strong feature extraction capabilities, and UNet was likely to suffer from overfitting problems depending on data characteristics or learning settings, resulting in poor performance. It is important to perform segmentation tasks considering the characteristics of each model and its suitability to the dataset.

Conclusion

Summary

Through this project, we wanted to find out which of the various CNN models do well in classification or segmentation when learning ultrasound images of breast cancer. Each of the three models has different ways of classifying and learningdata, and they have been found through investigations, testing, and analysis.

We expected UNet to show higher accuracy than FCN, but FCN performed better than UNet in both classification and segmentation. We predicted that there was an overfitting problem in the UNet learning process because we conducted learning and testing with little data.

We carried out this project to help to make a diagnosis with medical pictures. Since this diagnosis result is directly related to life, we paid attention to Precision, which is the value that is 'true among the predictions of the model'. Therefore, when classification and segmentation were performed among these three models, it was concluded that Yolov8, which has the highest precision, was suitable for analyzing ultrasound images of breast cancer.

Improvement

It can be seen that the performance of yolov8 is good, considering that the probability of accuracy and recall of yolov8 as well as precision is significantly higher than that of other models, but when segmentation is performed, yolov8 accuracy and recall are 61%, which is not high, so it is necessary to find a CNN model that is more suitable for breast cancer ultrasound images. In addition, it is necessary to secure more relevant ultrasound photographic data because the small amount of test data caused overfitting when learning UNet. Lastly, to compare and analyze the result of the stage of cancer, we need more accurate data that includes the ground truth of the stage of cancer.

Appendix

Code for Yolo

1. Train

   https://github.com/JinGaram/DLIP/blob/master/Yolo/Yolo_train.py

2. Test - 9 random

   https://github.com/JinGaram/DLIP/blob/master/Yolo/Yolo_test.py

3. Evaluation - 2 kinds

- Segmentaion Area Evaluation
  https://github.com/JinGaram/DLIP/blob/master/Yolo/Yolo_area_evaluation.py
- Post Image Evaluation
  https://github.com/JinGaram/DLIP/blob/master/Yolo/Yolo_6_post_evaluation.py

4. Trained Model

   https://github.com/JinGaram/DLIP/blob/master/Yolo/best.pt

   https://github.com/JinGaram/DLIP/blob/master/Yolo/yolov8s-seg.pt

5. Else

   https://github.com/JinGaram/DLIP/blob/master/Yolo/Yolo_6_predict_origin.py

Code for UNet

1. Train

   https://github.com/JinGaram/DLIP/blob/master/Unet/UNet_train.py

2. Test - 9 random

   https://github.com/JinGaram/DLIP/blob/master/Unet/UNet_test.py

3. Evaluation - 3 kinds

- Segmentaion Area Evaluation

  https://github.com/JinGaram/DLIP/blob/master/Unet/UNet_area_evaluation.py

- Post Image Evaluation

  https://github.com/JinGaram/DLIP/blob/master/Unet/UNet_post_evaluation.py

- Binary Evaluation

  https://github.com/JinGaram/DLIP/blob/master/Unet/UNet_binary_evaluation.py

4. Trained Model

   https://drive.google.com/file/d/1wJ_aw1VNJ8OgB63E1UITVD6irjBR-6l8/view?usp=drive_link

5. Else

   https://github.com/JinGaram/DLIP/blob/master/Unet/UNet_predict_origin.py

Code for FCN

1. Train

   https://github.com/JinGaram/DLIP/blob/master/FCN/FCN_train.py

2. Test - 9 random

   https://github.com/JinGaram/DLIP/blob/master/FCN/FCN_test.py

3. Evaluation - 3 kinds

- Segmentaion Area Evaluation

  https://github.com/JinGaram/DLIP/blob/master/FCN/FCN_area_evaluation.py

- Post Image Evaluation

  https://github.com/JinGaram/DLIP/blob/master/FCN/FCN_post_evaluation.py

- Binary Evaluation

  https://github.com/JinGaram/DLIP/blob/master/FCN/FCN_binary_evaluation.py

4. Trained Model

   https://drive.google.com/file/d/1r3vOr8_78HFJUAX9LB2PzucqeUYwub9e/view?usp=drive_link

5. Else

   https://github.com/JinGaram/DLIP/blob/master/FCN/FCN_predict_origin.py

Datasets

https://www.kaggle.com/datasets/aryashah2k/breast-ultrasound-images-dataset

Reference

[The MathWorks, Inc.] https://kr.mathworks.com/discovery/convolutional-neural-network.html

[Ultralytics] https://docs.ultralytics.com/models/yolov8/

[Cornell Univercity] **U-Net: Convolutional Networks for Biomedical Image Segmentation** https://arxiv.org/abs/1505.04597

[FCN] https://wikidocs.net/147359

[FCN] https://kuklife.tistory.com/117?category=872135

[FCN] https://medium.com/@msmapark2/fcn-논문-리뷰-fully-convolutional-networks-for-semantic-segmentation-81f016d76204

http://m.biospectator.com/view/news_view.php?varAtcId=19830

[Breast Cancer] https://www.kbcf.or.kr/bhi/bhi_info/present/death_rate_foreign.do

[Yolo] https://www.datacamp.com/blog/yolo-object-detection-explained