Breast Cancer Segmentation

Megan Rottkamp, David Gomez Camargo, Zuhal Tas-Batirbek

Links to our google colab notebooks

- Video Demo
- SAM
- MedSAM
- U-Net
- VGG-16

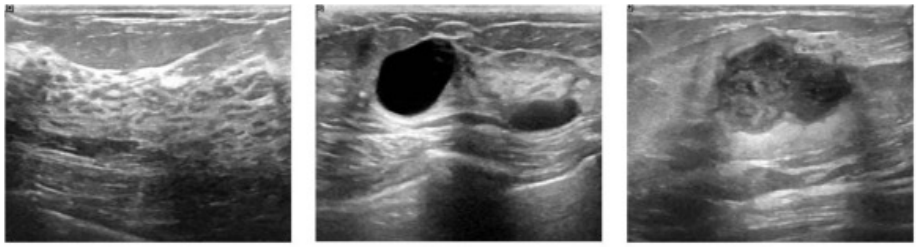
Our project's focus is to identify the breast tumor region in medical scans such as mammograms and ultrasound images, based on this example from Hugging Face. The project uses the Segment Anything Model (SAM).

One goal of our project will be to customize the model further by testing different hyper-parameters and techniques (learning rate, weight decay, number of epochs, activation function, etc) to improve accuracy/recall. Another goal of our project will be to experiment with two other techniques. One option is to use U-Net for the segmentation. U-Net is a popular deep-learning architecture for semantic segmentation. Originally developed for medical images. The other option is to use VGG-16 for image classification which is developed for increasing the depth of the network by adding more convolutional layers, which are the smallest possible layers.

Breast Ultrasound Images Dataset

We have found a kaggle dataset (official dataset page can be found here) that we will use for our implementation of SAM, U-Net, and VGG-16. The dataset contains 780 breast ultrasound images collected in 2018 from 600 female patients that are of age 25 to 75. Each image is classified as normal, benign, or malignant. Each image also has an associated masked image.

Case	Number of images
Benign	487
Malignant	210
Normal	133
Total	780



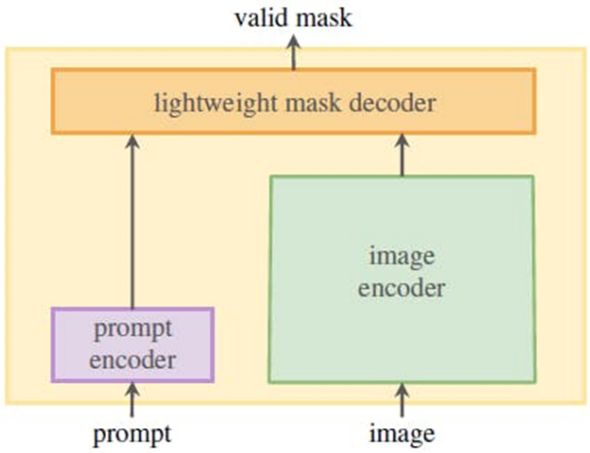
Normal Benign Malignant

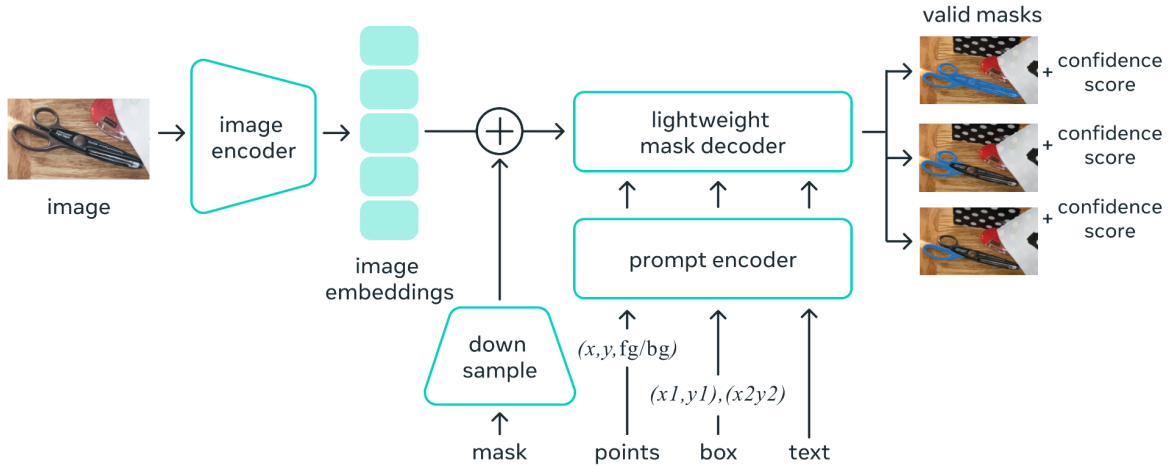
SAM

Official paper page:
https://scontent-lga3-2.xx.fbcdn.net/v/t39.2365-6/10000000_900554171201033_1602411987825904100_n.pdf?_nc_cat=100&ccb=1-7&_nc_sid=3c67a6&_nc_ohc=jt5d_yVzkZYAX-SFK5c&_nc_ht=scontent-lga3-2.xx&oh=00_AfB7pBqJ6jWxvHWLtaC5yyrgGrxpOmwlJpCfrhZg-PL5kA&oe=658729A7

The Segment Anything Model (SAM) is a new state-of-the-art image segmentation model developed and published by Meta AI in April, 2023. It can segment objects in an image or draw bounding boxes around objects. It was developed by Meta AI, who also developed Pytorch. SAM uses prompt engineering by taking input prompts such as segmenting images and then generates masks for object detection.

The architecture has an image encoder, a prompt encoder, and a lightweights mask decoder. The image encoder generates one-time image embeddings and the prompt encoder embeds the prompt. The lightweights mask decoder combines both of those embeddings and maps an output token to a mask. It uses self-attention and cross-attention. The masks are then used to update the model weights which allows the model to continue learning and improving. There is also a data engine that powers the tasks and builds the dataset (Segment Anything 1 Billion Mask). It has three stages: assisted-manual, semi-automatic, and fully automatic.





SAM was trained on millions of images over a billion masks. The dataset is currently the largest labeled segmentation dataset. There was a focus on the dataset having strong diversity to cover a wide range of domains, objects, and scenarios from various sources to ensure the model can generalize well. If the dataset is under-represented for a given task, then we can fine-tune the model weights.

Key Features of SAM

- **Zero-shot generalization**: SAM can be used to segment objects that it has never seen before, without the need for additional training.
- **Flexible prompting**: SAM can be prompted with a variety of input, including points, boxes, and text descriptions.

- **Real-time mask computation**: SAM can generate masks for objects in real time. This makes SAM ideal for applications where it is necessary to segment objects quickly, such as autonomous driving and robotics.
- **Ambiguity awareness**: SAM is aware of the ambiguity of objects in images. This means that SAM can generate masks for objects even when they are partially occluded or overlapping with other objects.

  Below are snippets of code for our own implementation of SAM:

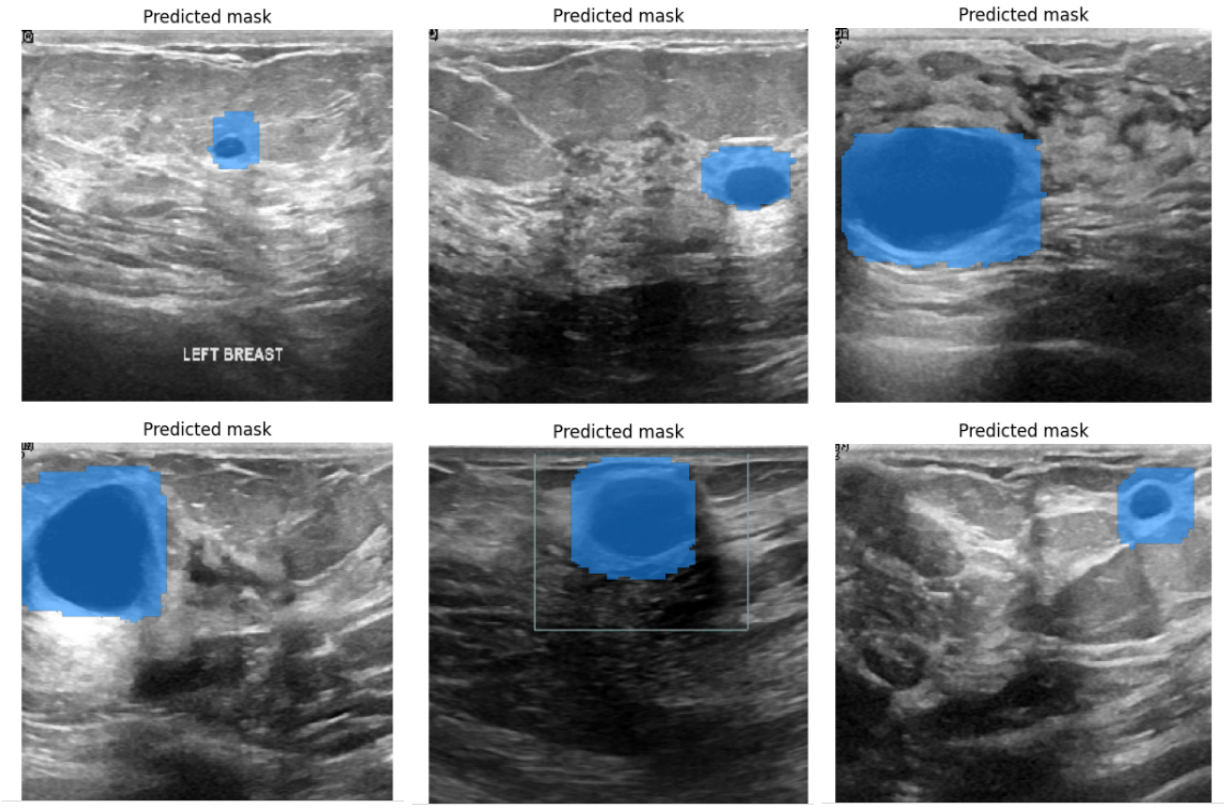
Training:

```
from transformers import SamProcessor

processor = SamProcessor.from_pretrained("facebook/sam-vit-base")
```

```
from torch.optim import Adam
import monai

optimizer = Adam(model.mask_decoder.parameters(), lr=1e-5, weight_decay=0)

seg_loss = monai.losses.DiceCELoss(sigmoid=True, squared_pred=True, reduction='mean')
```

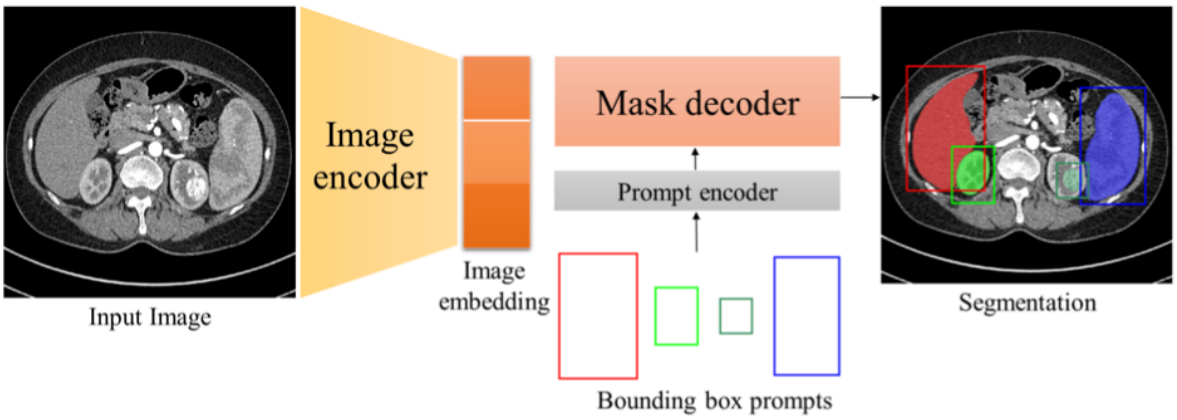
```
num_epochs = 20

device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)

model.train()
for epoch in range(num_epochs):
    epoch_losses = []
    for batch in tqdm(train_dataloader):
      # forward pass
      outputs = model(pixel_values=batch["pixel_values"].to(device),
                      input_boxes=batch["input_boxes"].to(device),
                      multimask_output=False)

      # compute loss
      predicted_masks = outputs.pred_masks.squeeze(1)
      ground_truth_masks = batch["ground_truth_mask"].float().to(device)
      loss = seg_loss(predicted_masks, ground_truth_masks.unsqueeze(1))

      # backward pass (compute gradients of parameters w.r.t. loss)
      optimizer.zero_grad()
      loss.backward()

      # optimize
      optimizer.step()
      epoch_losses.append(loss.item())

    print(f'EPOCH: {epoch}')
    print(f'Mean loss: {mean(epoch_losses)}')
```

Sample Output:

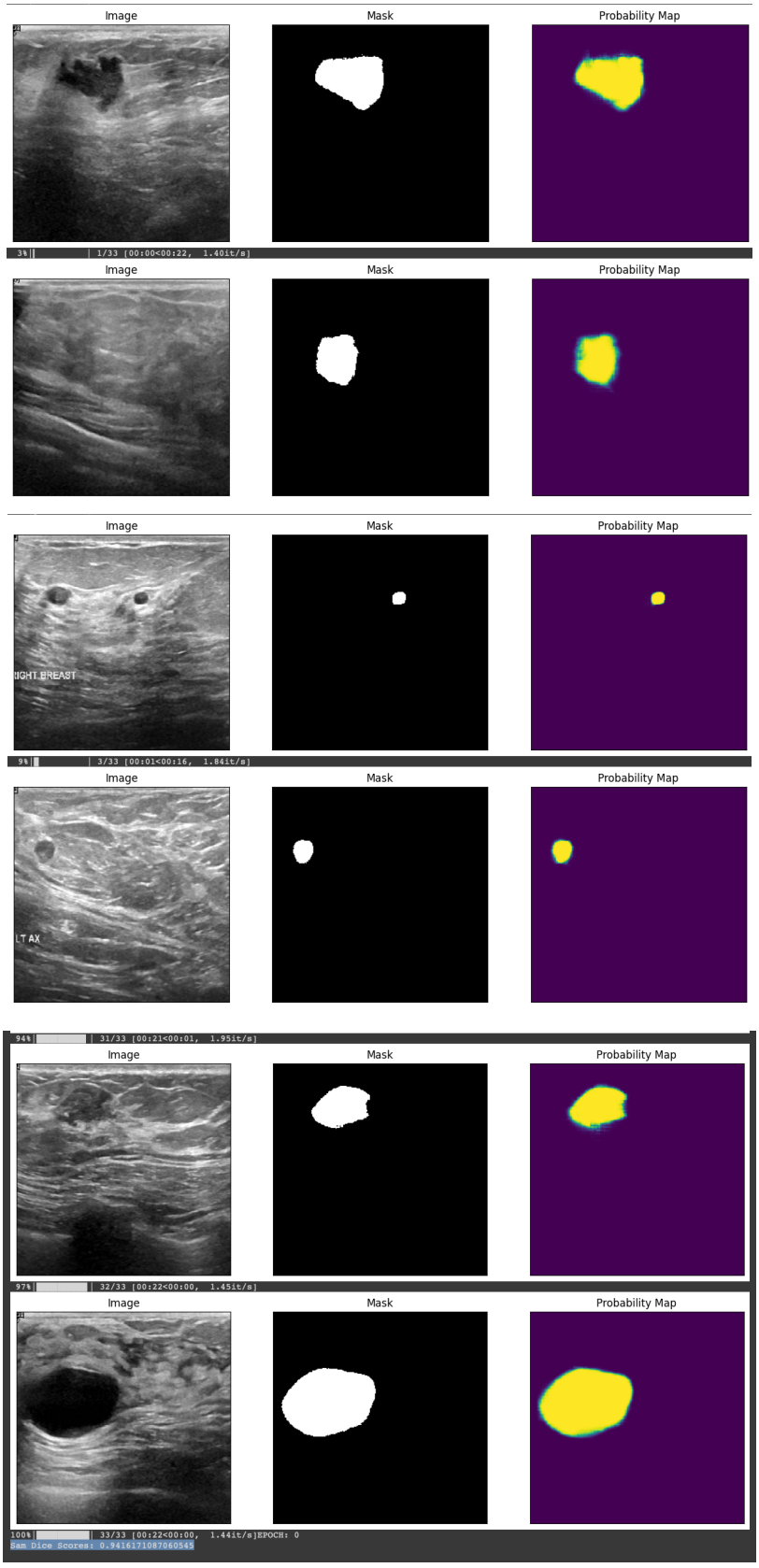


MedSAM

MedSAM was introduced shortly after the SAM paper was released. Writers claim MedSAM to be the first foundation model for universal medical image segmentation. To meet this challenge, they curated a diverse and large-scale medical image segmentation dataset with 1,090,486 medical image-mask pairs, over 30 cancer types. This large-scale dataset allows MedSAM to learn a rich representation of medical images.



Sample Output of MedSAM:



U-Net

UNet is a popular architecture in deep learning, specifically designed for semantic segmentation tasks, and it has found widespread use in the medical field. It was originally proposed by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015. UNet is particularly well-suited for tasks where precise localization and delineation of structures in images are crucial, such as medical image segmentation.
Here's a brief description of the UNet architecture:

Encoder-Decoder Structure:

- UNet follows an encoder-decoder architecture. The encoder part captures the context and extracts features from the input image, while the decoder part helps in localization and segmentation.

Contracting Path (Encoder):

- The encoder consists of multiple convolutional and pooling layers arranged in a contracting path. Each convolutional layer is typically followed by batch normalization and a rectified linear unit (ReLU) activation function. Pooling layers are used to reduce the spatial dimensions of the input and increase the receptive field.

Bottleneck:

- At the bottom of the network is a bottleneck layer that captures the most essential information from the input image. This layer is often composed of multiple convolutional layers.

Expanding Path (Decoder):

- The decoder is the expanding path of the network, and it consists of up-sampling and convolutional layers. The up-sampling helps restore the spatial dimensions of the feature maps.

Skip Connections:

- One key innovation in UNet is the use of skip connections between the encoder and decoder. These connections concatenate feature maps from the encoder to the corresponding layer in the decoder. This allows the network to retain high-resolution information during the upsampling process and helps in precise localization.

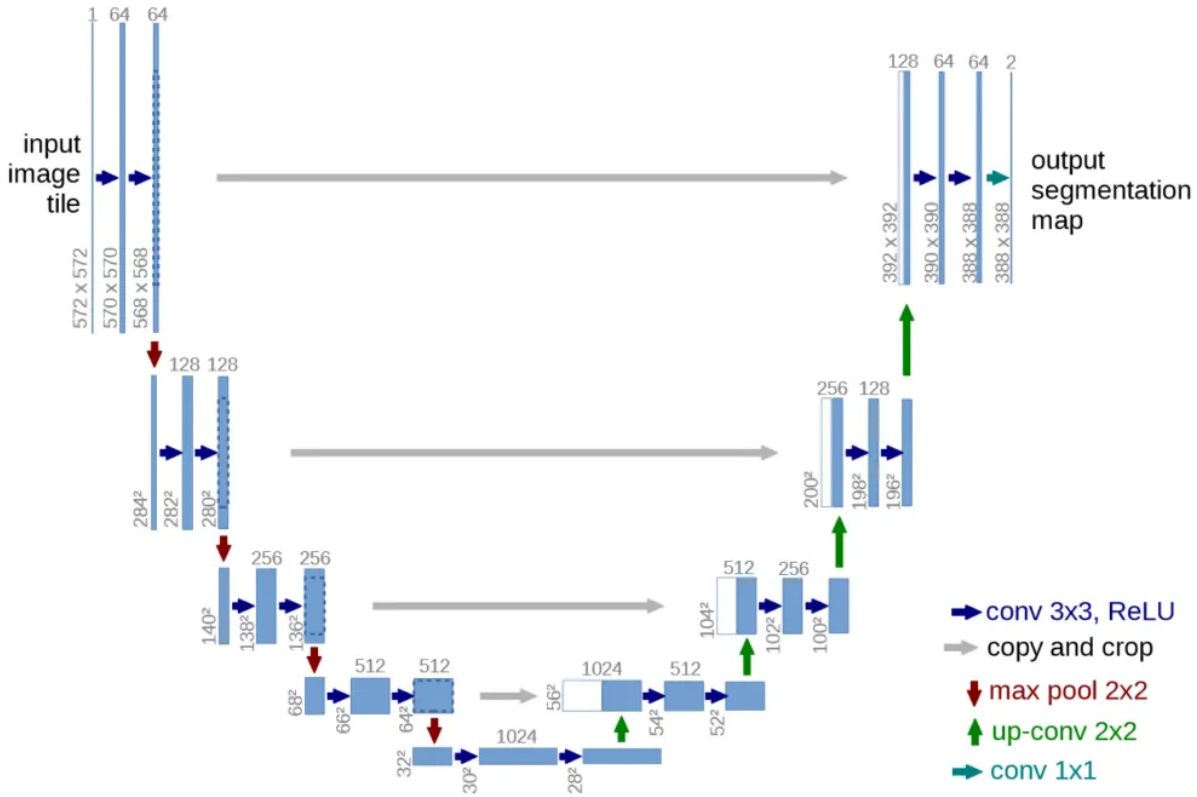
Final Layer:

- The final layer typically consists of a 1x1 convolutional layer followed by a softmax activation function. This layer produces a segmentation map where each pixel is assigned a class label, indicating the presence or absence of a particular structure in the image.

In the medical field, UNet has been extensively used for tasks such as:

- Medical Image Segmentation: Identifying and delineating structures of interest in medical images, such as tumors, organs, or blood vessels.
- Biomedical Image Analysis: Analyzing and extracting features from various biomedical images for diagnostic or research purposes.
- Image-to-Image Translation: Converting one type of medical image to another, for example, from CT to MRI or from low-resolution to high-resolution images.

The UNet architecture's ability to capture intricate details and localize structures effectively makes it a valuable tool in medical image analysis. Its success has led to numerous variations and improvements, and it continues to be a go-to architecture in the field.



Below are snippets of code for our own implementation of U-Net:

[15]
```
# Contract Path

inply = Input((128, 128, 1,))

conv1 = Conv2D(2**6, (3,3), activation = 'relu', padding = 'same')(inply)
conv1 = Conv2D(2**6, (3,3), activation = 'relu', padding = 'same')(conv1)
pool1 = MaxPooling2D((2,2), strides = 2, padding = 'same')(conv1)
drop1 = Dropout(0.2)(pool1)

conv2 = Conv2D(2**7, (3,3), activation = 'relu', padding = 'same')(drop1)
conv2 = Conv2D(2**7, (3,3), activation = 'relu', padding = 'same')(conv2)
pool2 = MaxPooling2D((2,2), strides = 2, padding = 'same')(conv2)
drop2 = Dropout(0.2)(pool2)

conv3 = Conv2D(2**8, (3,3), activation = 'relu', padding = 'same')(drop2)
conv3 = Conv2D(2**8, (3,3), activation = 'relu', padding = 'same')(conv3)
pool3 = MaxPooling2D((2,2), strides = 2, padding = 'same')(conv3)
drop3 = Dropout(0.2)(pool3)

conv4 = Conv2D(2**9, (3,3), activation = 'relu', padding = 'same')(drop3)
conv4 = Conv2D(2**9, (3,3), activation = 'relu', padding = 'same')(conv4)
pool4 = MaxPooling2D((2,2), strides = 2, padding = 'same')(conv4)
drop4 = Dropout(0.2)(pool4)
```

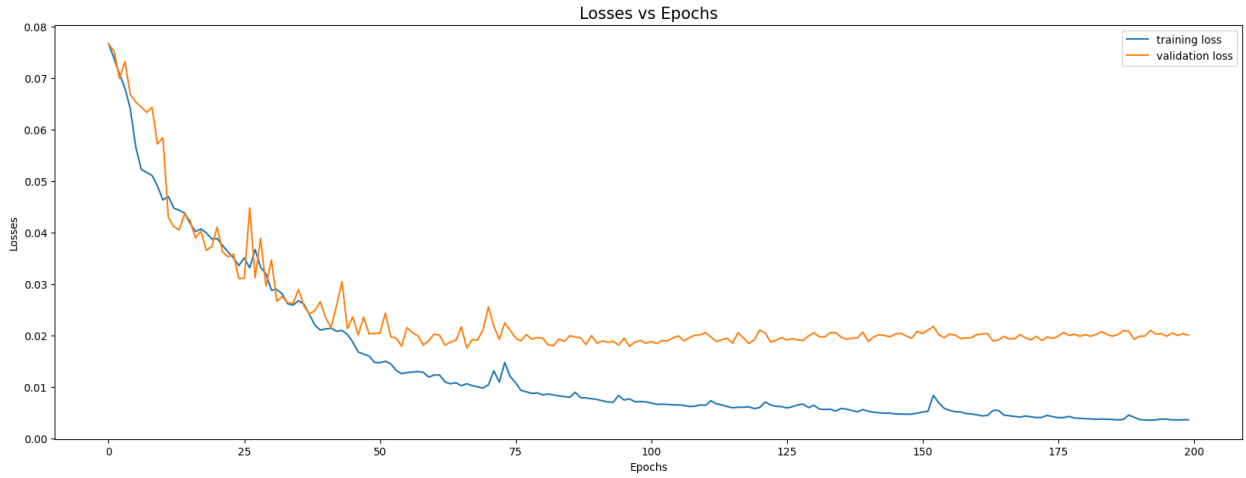
[16]
```
# Bottleneck Layer

convm = Conv2D(2**10, (3,3), activation = 'relu', padding = 'same')(drop4)
convm = Conv2D(2**10, (3,3), activation = 'relu', padding = 'same')(convm)
```

[17]
```
# Expanding Layer

tran5 = Conv2DTranspose(2**9, (2,2), strides = 2, padding = 'valid', activation = 'relu')(convm)
conc5 = Concatenate()([tran5, conv4])
conv5 = Conv2D(2**9, (3,3), activation = 'relu', padding = 'same')(conc5)
conv5 = Conv2D(2**9, (3,3), activation = 'relu', padding = 'same')(conv5)
drop5 = Dropout(0.1)(conv5)

tran6 = Conv2DTranspose(2**8, (2,2), strides = 2, padding = 'valid', activation = 'relu')(drop5)
conc6 = Concatenate()([tran6, conv3])
conv6 = Conv2D(2**8, (3,3), activation = 'relu', padding = 'same')(conc6)
conv6 = Conv2D(2**8, (3,3), activation = 'relu', padding = 'same')(conv6)
drop6 = Dropout(0.1)(conv6)

tran7 = Conv2DTranspose(2**7, (2,2), strides = 2, padding = 'valid', activation = 'relu')(drop6)
conc7 = Concatenate()([tran7, conv2])
conv7 = Conv2D(2**7, (3,3), activation = 'relu', padding = 'same')(conc7)
conv7 = Conv2D(2**7, (3,3), activation = 'relu', padding = 'same')(conv7)
drop7 = Dropout(0.1)(conv7)

tran8 = Conv2DTranspose(2**6, (2,2), strides = 2, padding = 'valid', activation = 'relu')(drop7)
conc8 = Concatenate()([tran8, conv1])
conv8 = Conv2D(2**6, (3,3), activation = 'relu', padding = 'same')(conc8)
conv8 = Conv2D(2**6, (3,3), activation = 'relu', padding = 'same')(conv8)
drop8 = Dropout(0.1)(conv8)
```

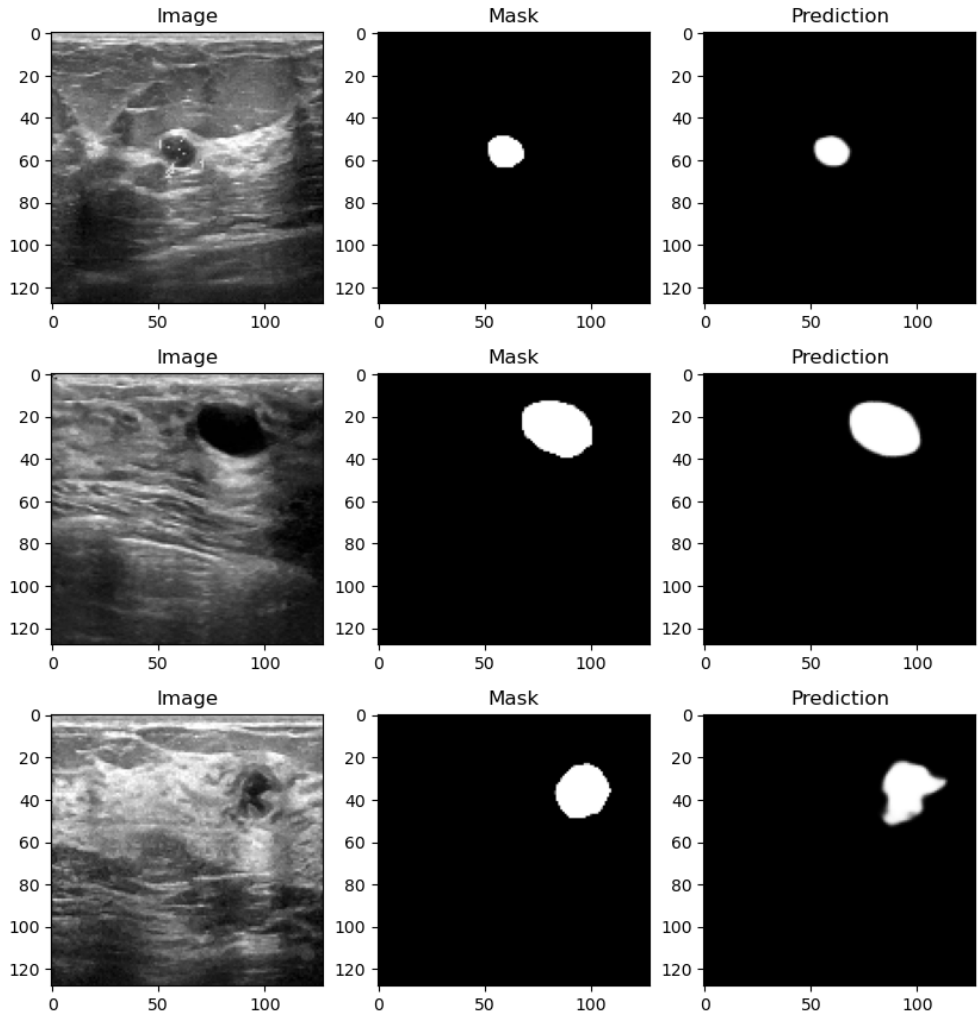
[18]
```
outly = Conv2D(2**0, (1,1), activation = 'relu', padding = 'same')(drop8)
model = Model(inputs = inply, outputs = outly, name = 'U-net')
```

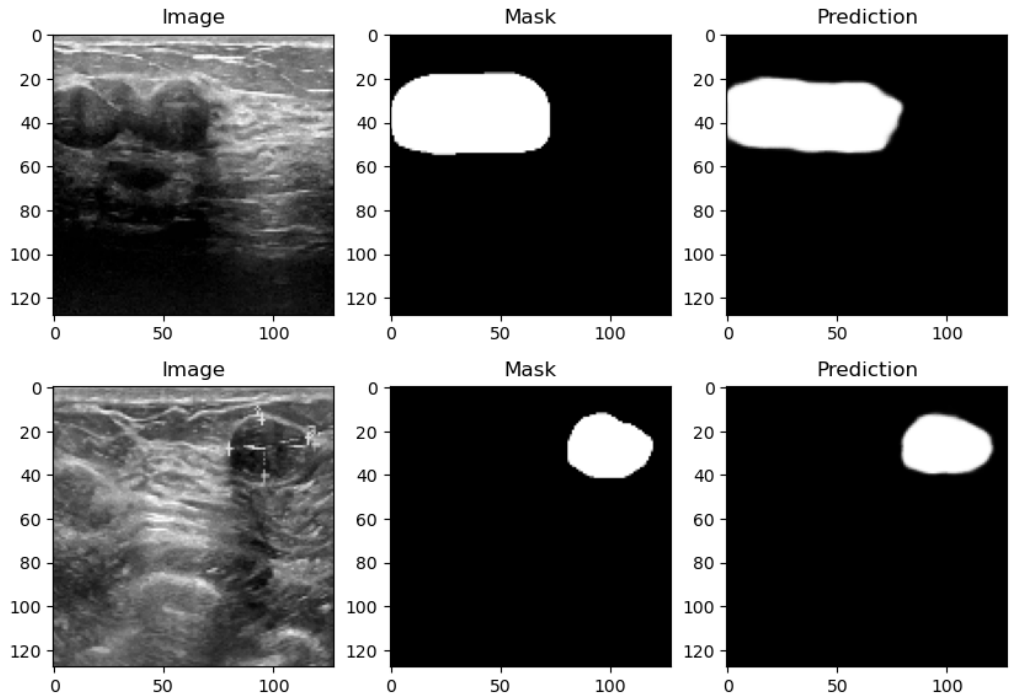
[19]
```
# Loss Function

from keras.metrics import MeanIoU
```

Model performance:



Results:



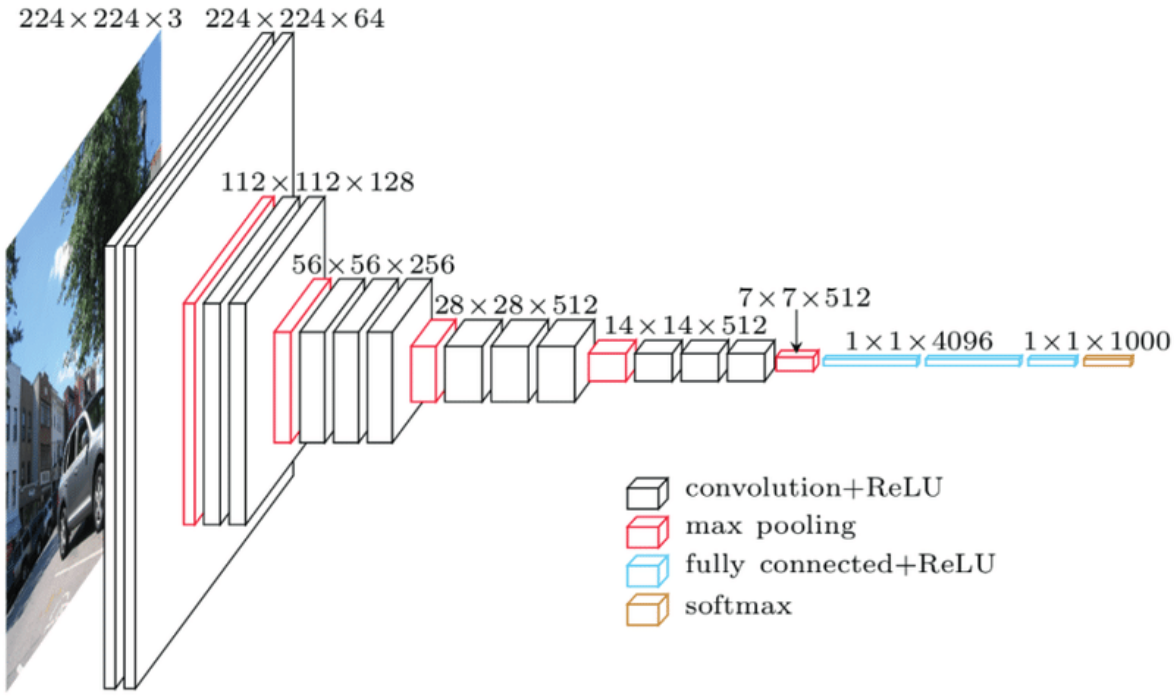


VGG-16

VGG-16 is a convolution neural network (CNN) model supporting 16 layers. K. Simonyan and A. Zisserman from Oxford University proposed this model and published it in a paper in 2014. This model differed from previous high-performing models in several ways;

First, it used a tiny 3×3 receptive field with a 1-pixel stride. The benefit of using multiple smaller layers rather than a single large layer is that more non-linear activation layers accompany the convolution layers, improving the decision functions and allowing the network to converge quickly.

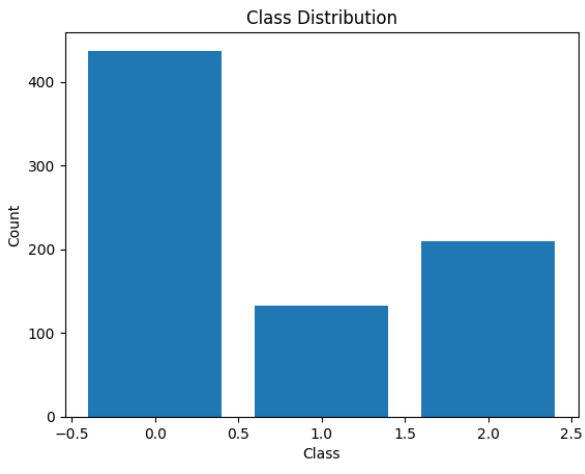
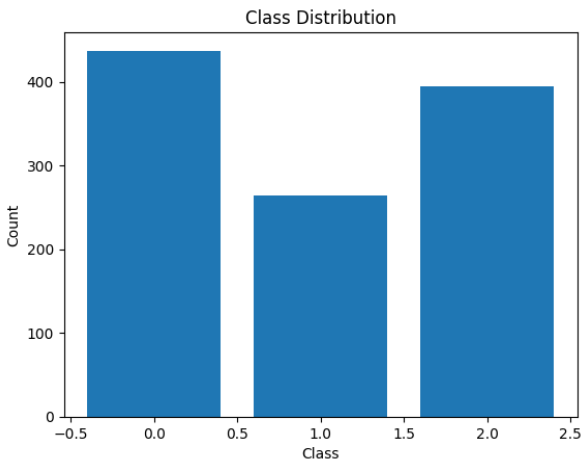
Second, VGG uses a smaller convolutional filter, which reduces the network's tendency to over-fit during training exercises. A 3×3 filter is the optimal size because a smaller size cannot capture left-right and up-down information. Thus, VGG is the smallest possible model to understand an image's spatial features. Consistent 3×3 convolutions make the network easy to manage.



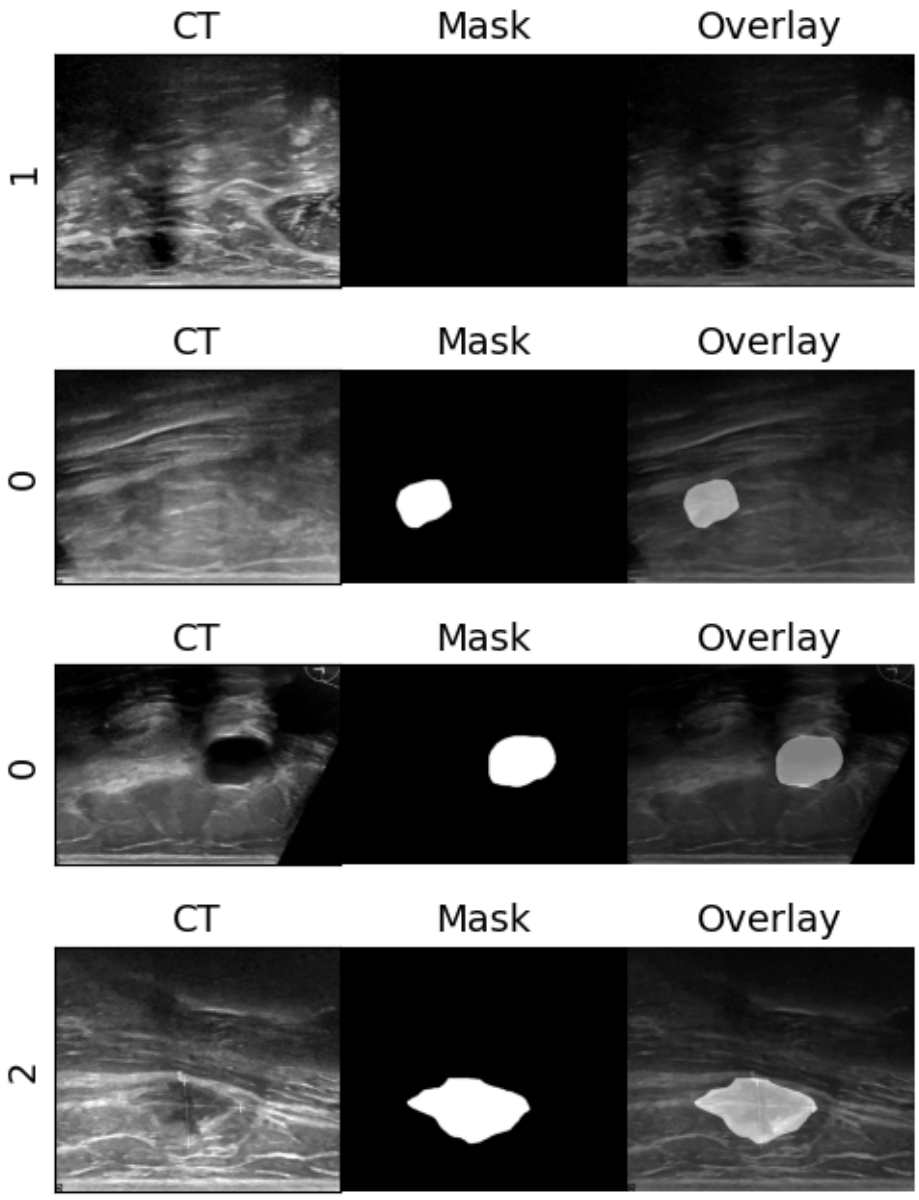
VGG16 has three fully connected layers and 13 convolutional layers.

To improve the classification accuracy while using VGG-16, different methods are used:

1. Data Augmentation:

Before Data Augmentation	After Data Augmentation
	

2. Overlaying Images and Masks



Results of VGG Model:

Training and Test Accuracies

```
Epoch 40/50 => Train Loss: 0.1011, Train Accuracy: 0.9765 | Val Loss: 0.3209, Val Accuracy: 0.8963
Epoch 41/50 => Train Loss: 0.0687, Train Accuracy: 0.9830 | Val Loss: 0.3164, Val Accuracy: 0.9146
Epoch 42/50 => Train Loss: 0.0754, Train Accuracy: 0.9830 | Val Loss: 0.2576, Val Accuracy: 0.9085
Epoch 43/50 => Train Loss: 0.0835, Train Accuracy: 0.9778 | Val Loss: 0.3606, Val Accuracy: 0.8780
Epoch 44/50 => Train Loss: 0.0737, Train Accuracy: 0.9765 | Val Loss: 0.3555, Val Accuracy: 0.9085
Epoch 45/50 => Train Loss: 0.0794, Train Accuracy: 0.9778 | Val Loss: 0.4804, Val Accuracy: 0.8902
Epoch 46/50 => Train Loss: 0.0462, Train Accuracy: 0.9791 | Val Loss: 0.5127, Val Accuracy: 0.9085
Epoch 47/50 => Train Loss: 0.0509, Train Accuracy: 0.9830 | Val Loss: 0.4443, Val Accuracy: 0.9146
Epoch 48/50 => Train Loss: 0.0751, Train Accuracy: 0.9752 | Val Loss: 0.4187, Val Accuracy: 0.8963
Epoch 49/50 => Train Loss: 0.1116, Train Accuracy: 0.9700 | Val Loss: 0.2788, Val Accuracy: 0.9329
Epoch 50/50 => Train Loss: 0.0672, Train Accuracy: 0.9817 | Val Loss: 0.6191, Val Accuracy: 0.8780
```

```python
# Test on Validation Set
correct_predictions = 0
total_samples = 0
with torch.no_grad():
    for inputs, labels in val_dataloader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        correct_predictions += (predicted == labels).sum().item()
        total_samples += labels.size(0)

# Calculate accuracy
accuracy = correct_predictions / total_samples
print(f'Val Accuracy: {accuracy * 100:.2f}%')
```

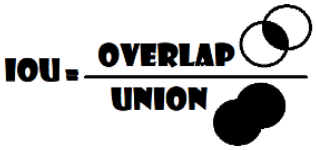
```
Val Accuracy: 86.06%
```

Evaluation Metrics for Semantic Segmentation

When evaluating the performance of an image segmentation model, we need to compare the model's predicted masks with the ground truth masks to quantify how accurately the model identifies regions or boundaries of the object within the image.

1. Intersection over Union (IoU) or Jaccard Index

IoU measures the overlap between the predicted mask and the ground truth mask. It provides a measure of how well the model's predictions align with the actual regions of interest in the images.

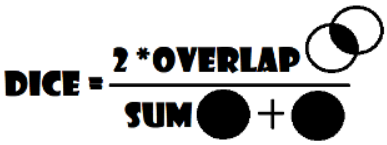


$$Jaccard(A, B) = \frac{\|A \cap B\|}{\|A \cup B\|} \qquad Jaccard = IoU = \frac{TP}{TP + FP + FN}$$

It's computed as the ratio of the intersection area, where both the predicted and ground truth masks have non-zero values, to the union area, where either the predicted or ground truth mask has non-zero values.

2. Dice Coefficient or F1 Score

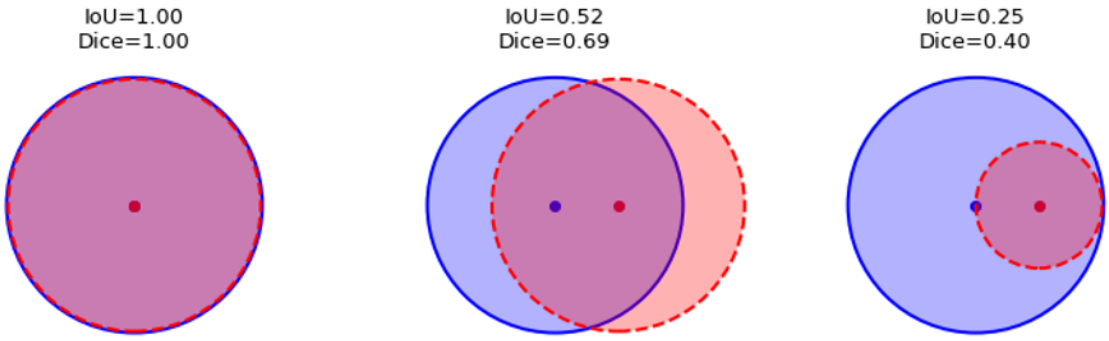
Dice coefficient is another measure of how well the model's predictions match the ground truth, emphasizing the overlap between the two masks. Dice coefficient quantifies the similarity between the predicted mask and the ground truth mask.



$$Dice(A, B) = \frac{2\|A \cap B\|}{\|A\| + \|B\|} \quad Dice = \frac{2TP}{2TP + FP + FN}$$

It's computed as twice the intersection of the predicted and ground truth masks divided by the total number of pixels in both masks.

Dice Coefficient value is between 0 and 1; 1 indicates a perfect overlap while 0 indicates no overlap. Here is an illustration of the Dice and IoU metrics given two circles representing the ground truth and the predicted masks for an arbitrary object class:



We used Dice Loss for our SAM models.Dice loss and dice coefficients are not the same things:

Dice Loss = 1 - Dice Coefficient

Our SAM model's best dice (f1) score is: 1-0.064 = **0.936,** which is pretty close to 1.

3. Pixel-Wise Accuracy

Pixel-wise accuracy calculates the percentage of correctly classified pixels. Pixel-wise accuracy gives an overall idea of how accurately the model predicts individual pixels in the segmentation masks.

It's computed by counting the number of pixels where the predicted mask matches the ground truth mask and dividing it by the total number of pixels in the mask.

Pixel-wise accuracy method is generally not ideal when you have an imbalanced dataset. Handling class imbalance is a crucial step before using this method.

Our Models and their Training and Validation Losses:

Model	**Loss Function**	**Training Loss**	**Validation Loss**
U-Net	Mean Squared Error	0.0036	0.0200
SAM	Dice Loss	0.070	0.067
MedSAM	Dice Loss	0.064	0.058
VGG-16	Cross Entropy Loss	0.0672	0.6191

REFERENCES

Official Page of Dataset
https://www.sciencedirect.com/science/article/pii/S2352340919312181

Official Repository for SAM
https://github.com/facebookresearch/segment-anything

SAM publication on Meta AI
https://ai.meta.com/research/publications/segment-anything/

Official SAM Research Paper
https://scontent-lga3-2.xx.fbcdn.net/v/t39.2365-6/10000000_900554171201033_1602411987825904100_n.pdf?_nc_cat=100&ccb=1-7&_nc_sid=3c67a6&_nc_ohc=jt5d_yVzkZYAX-SFK5c&_nc_ht=scontent-lga3-2.xx&oh=00_AfB7pBqJ6jWxvHWLtaC5yyrgGrxpOmwIJpCfrhZg-PL5kA&oe=658729A7

Official MedSAM Research Paper
https://arxiv.org/pdf/2304.12306.pdf

Official Repository for MedSAM
https://github.com/bowang-lab/MedSAM

Official VGG-16 Research Paper
https://arxiv.org/pdf/1409.1556.pdf

Helpful Notebooks & Tutorials
https://github.com/NielsRogge/Transformers-Tutorials/blob/master/SAM/Fine_tune_SAM_(segment_anything)_on_a_custom_dataset.ipynb

https://github.com/bnsreenu/python_for_microscopists/blob/master/331_fine_tune_SAM_mito.ipynb

https://github.com/NielsRogge/Transformers-Tutorials/blob/master/SAM/Run_inference_with_MedSAM_using_HuggingFace_Transformers.ipynb

https://www.kaggle.com/code/parsakh/breast-cancer-classification-vgg16

https://ilmonteux.github.io/2019/05/10/segmentation-metrics.html

https://pycad.co/the-difference-between-dice-and-dice-loss/#google_vignette