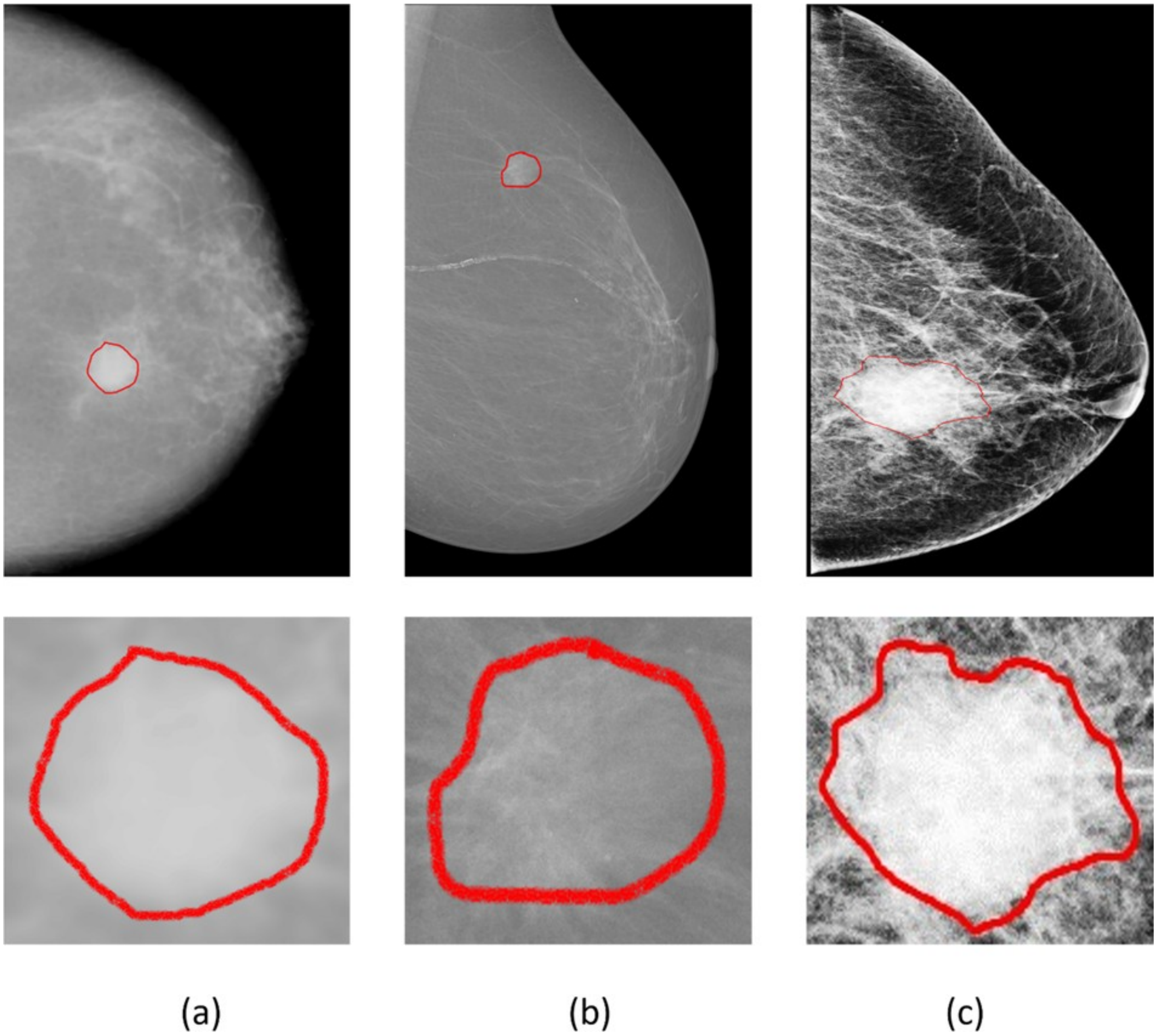
Breast Cancer Segmentation



```python
import numpy as np
import pandas as pd
import os

base_path =
'/kaggle/input/breast-ultrasound-images-dataset/Dataset_BUSI_with_GT/'

tumor_types = ["benign", "malignant", "normal"]

image_paths = []
mask_paths = []
tumor_labels = []
```

```
for tumor in tumor_types:
    folder_path = os.path.join(base_path, tumor)

    if os.path.exists(folder_path):

        files = os.listdir(folder_path)

        image_files = [f for f in files if f.endswith(".png") and
"_mask" not in f]

        for img_file in image_files:
            mask_file = img_file.replace(".png", "_mask.png")

            img_path = os.path.join(folder_path, img_file)
            mask_path = os.path.join(folder_path, mask_file)

            if os.path.exists(img_path) and os.path.exists(mask_path):
                image_paths.append(img_path)
                mask_paths.append(mask_path)
                tumor_labels.append(tumor)
            else:
                print(f"Missing pair for image: {img_path} or mask:
{mask_path}")
    else:
        print(f"Folder not found: {folder_path}")

df = pd.DataFrame({
    "image_path": image_paths,
    "mask_path": mask_paths,
    "tumor_type": tumor_labels
})

df
```

```
                                            image_path  \
0    /kaggle/input/breast-ultrasound-images-dataset...
1    /kaggle/input/breast-ultrasound-images-dataset...
2    /kaggle/input/breast-ultrasound-images-dataset...
3    /kaggle/input/breast-ultrasound-images-dataset...
4    /kaggle/input/breast-ultrasound-images-dataset...
..                                                 ...
775  /kaggle/input/breast-ultrasound-images-dataset...
776  /kaggle/input/breast-ultrasound-images-dataset...
777  /kaggle/input/breast-ultrasound-images-dataset...
778  /kaggle/input/breast-ultrasound-images-dataset...
779  /kaggle/input/breast-ultrasound-images-dataset...

                                             mask_path tumor_type
0    /kaggle/input/breast-ultrasound-images-dataset...     benign
1    /kaggle/input/breast-ultrasound-images-dataset...     benign
```

```
2    /kaggle/input/breast-ultrasound-images-dataset...     benign
3    /kaggle/input/breast-ultrasound-images-dataset...     benign
4    /kaggle/input/breast-ultrasound-images-dataset...     benign
..                                                 ...        ...
775  /kaggle/input/breast-ultrasound-images-dataset...     normal
776  /kaggle/input/breast-ultrasound-images-dataset...     normal
777  /kaggle/input/breast-ultrasound-images-dataset...     normal
778  /kaggle/input/breast-ultrasound-images-dataset...     normal
779  /kaggle/input/breast-ultrasound-images-dataset...     normal

[780 rows x 3 columns]
```

```
df.shape
```

```
(780, 3)
```

```
df.columns
```

```
Index(['image_path', 'mask_path', 'tumor_type'], dtype='object')
```

```
df.duplicated().sum()
```

```
0
```

```
df.isnull().sum()
```

```
image_path    0
mask_path     0
tumor_type    0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 780 entries, 0 to 779
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  780 non-null    object
 1   mask_path   780 non-null    object
 2   tumor_type  780 non-null    object
dtypes: object(3)
memory usage: 18.4+ KB
```

```
df['tumor_type'].unique()
```

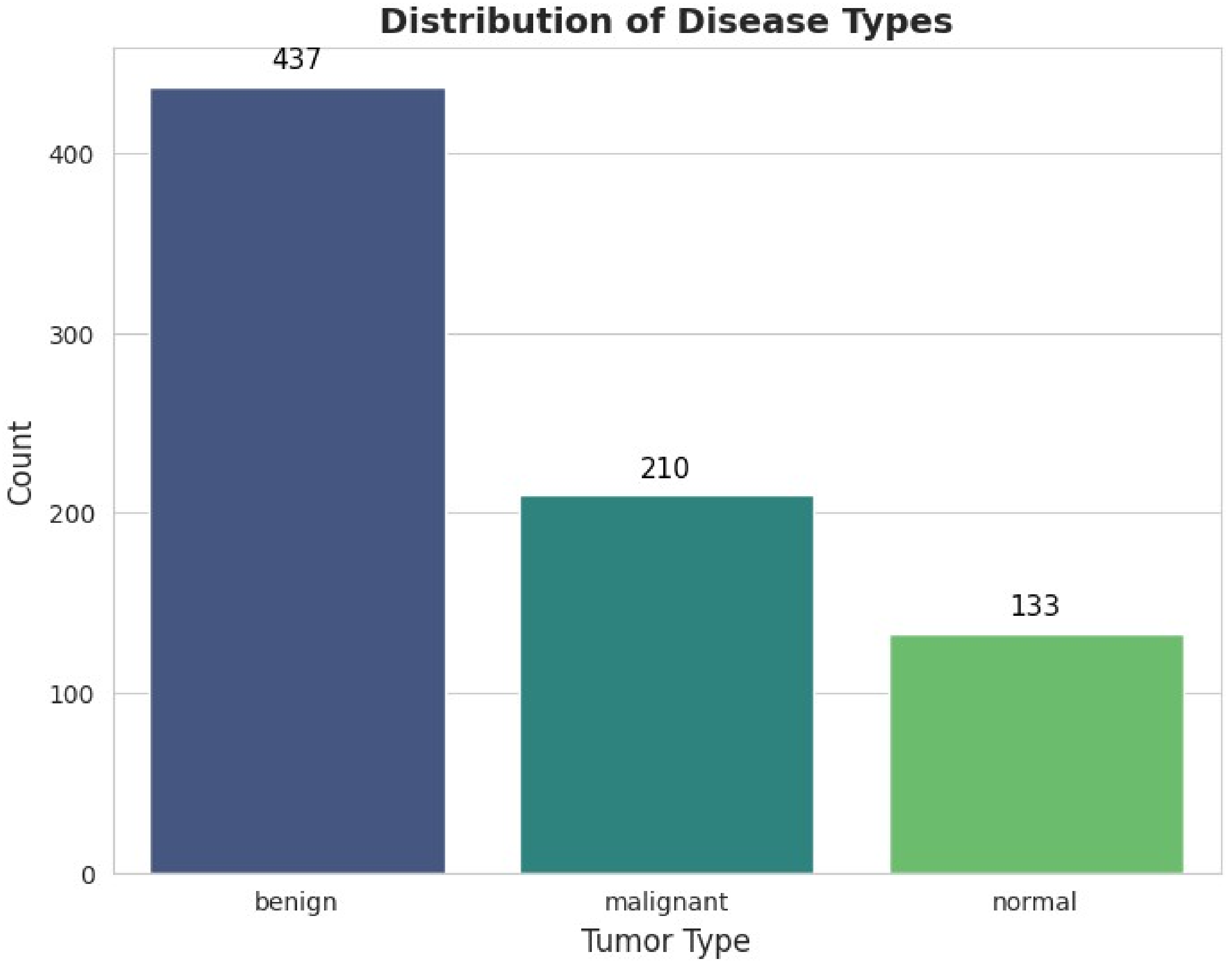
```
array(['benign', 'malignant', 'normal'], dtype=object)
```

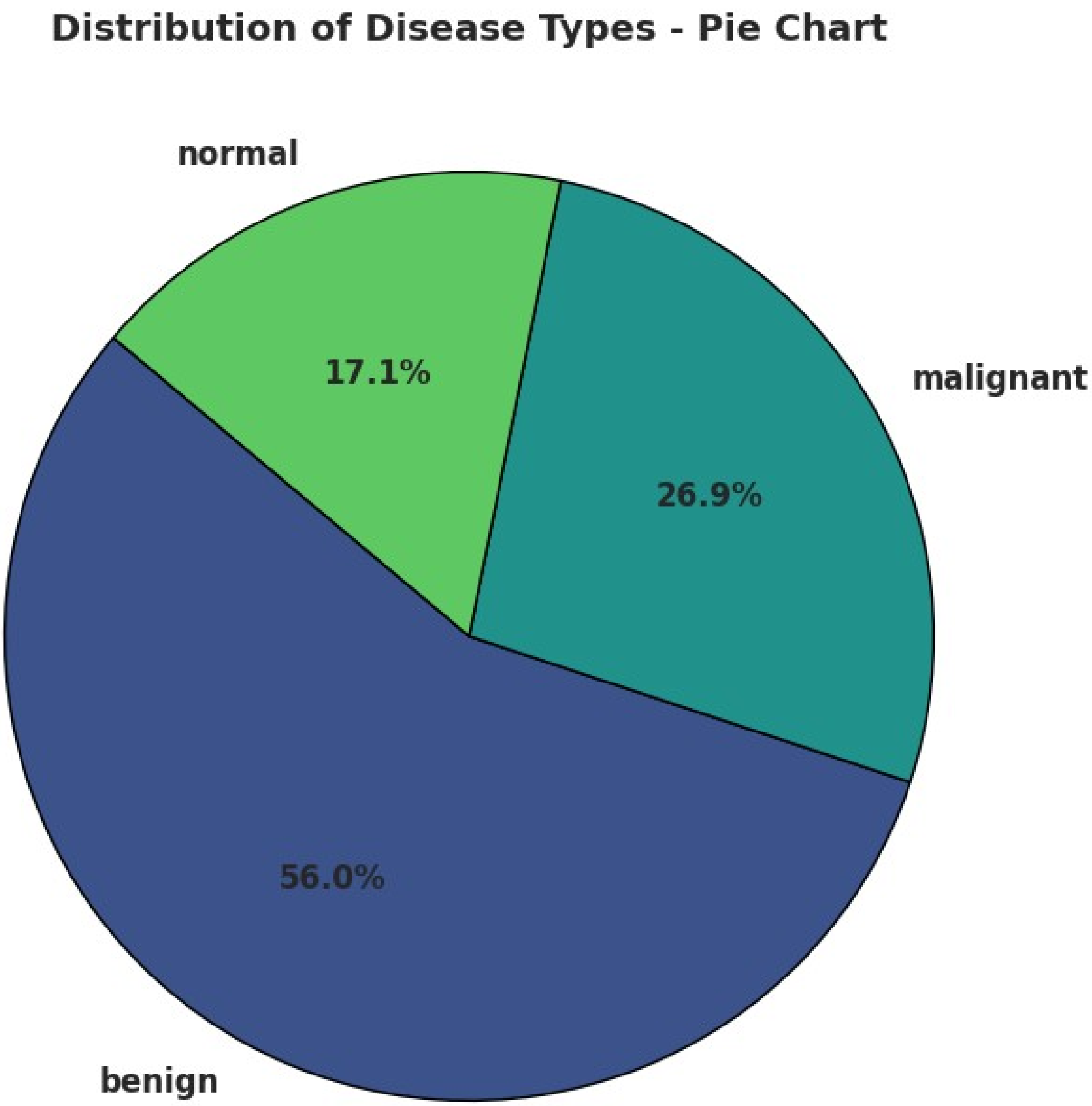
```
df['tumor_type'].value_counts()
```

```
tumor_type
benign       437
```

```
malignant    210
normal       133
Name: count, dtype: int64
```

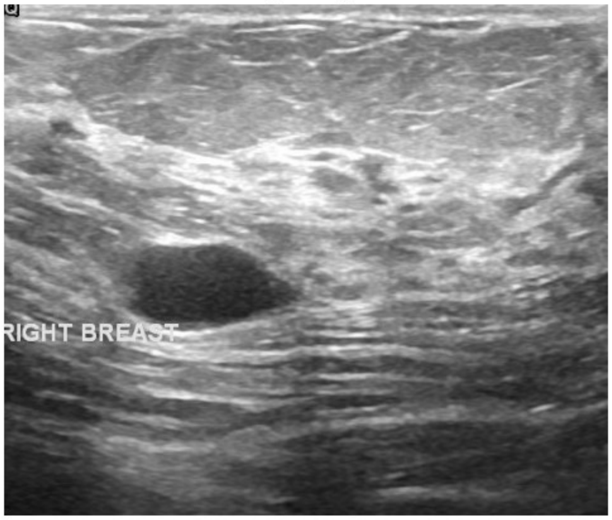
```python
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(8, 6))
sns.countplot(data=df, x="tumor_type", palette="viridis", ax=ax)

ax.set_title("Distribution of Disease Types", fontsize=14,
fontweight='bold')
ax.set_xlabel("Tumor Type", fontsize=12)
ax.set_ylabel("Count", fontsize=12)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=11, color='black',
                xytext=(0, 5), textcoords='offset points')

plt.show()

label_counts = df["tumor_type"].value_counts()

fig, ax = plt.subplots(figsize=(20, 8))
colors = sns.color_palette("viridis", len(label_counts))

ax.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%',
       startangle=140, colors=colors, textprops={'fontsize': 12,
'weight': 'bold'},
       wedgeprops={'edgecolor': 'black', 'linewidth': 1})

ax.set_title("Distribution of Disease Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution of Disease Types
437
210
133
400
300
200
100
0
Count
benign
malignant
normal
Tumor Type

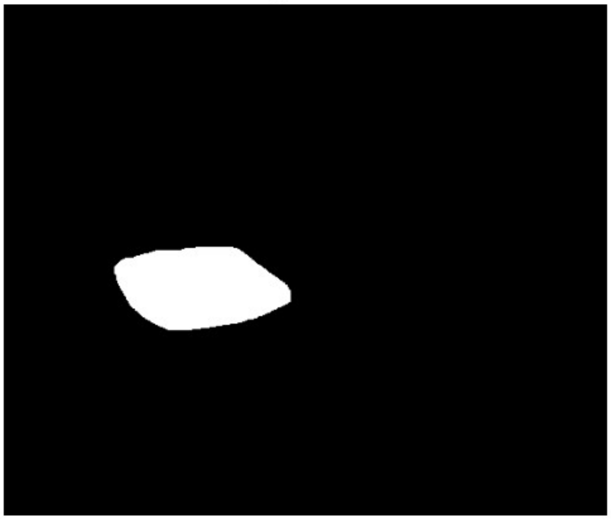


```
from PIL import Image

def display_images_and_masks(df, num_images=5):

    fig, axes = plt.subplots(nrows=len(tumor_types) * num_images,
ncols=2,
                             figsize=(10, 5 * len(tumor_types) *
num_images))

    for idx, tumor in enumerate(tumor_types):
        tumor_df = df[df["tumor_type"] == tumor].head(num_images)
        for i, (_, row) in enumerate(tumor_df.iterrows()):
            img = np.array(Image.open(row["image_path"]))
            mask = np.array(Image.open(row["mask_path"]))

            ax_img = axes[idx * num_images + i, 0]
            ax_img.imshow(img, cmap='gray' if len(img.shape) == 2 else None)
            ax_img.set_title(f"{tumor} Image {i+1}")
            ax_img.axis('off')

            ax_mask = axes[idx * num_images + i, 1]
            ax_mask.imshow(mask, cmap='gray')
            ax_mask.set_title(f"{tumor} Mask {i+1}")
            ax_mask.axis('off')

    plt.tight_layout()
    plt.show()

display_images_and_masks(df, num_images=5)
```

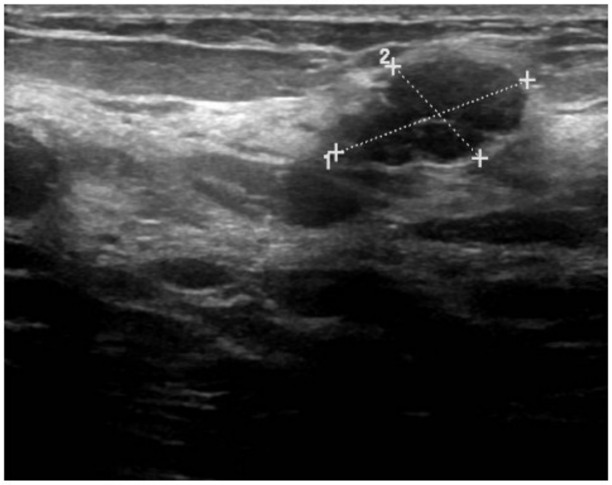
benign Image 1



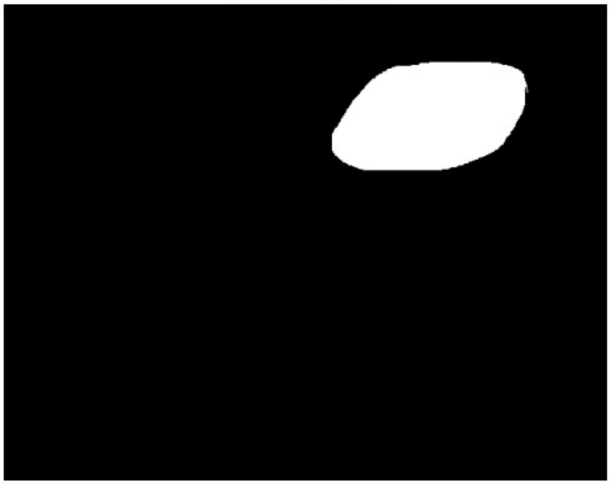
benign Mask 1



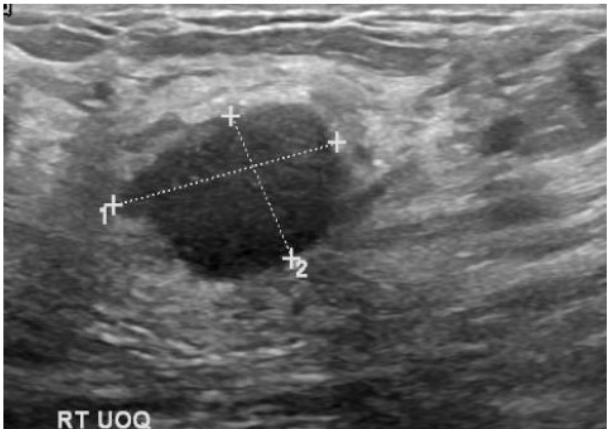
benign Image 2



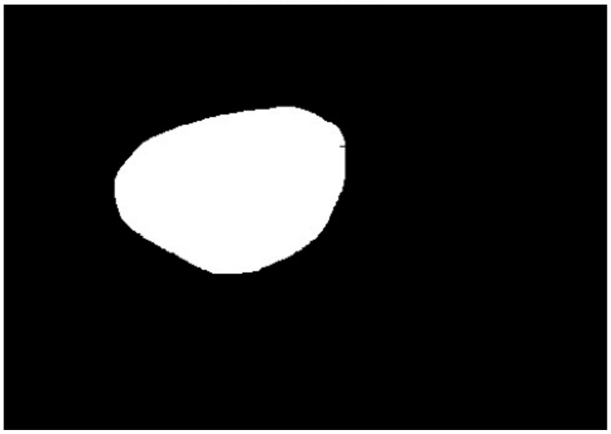
benign Mask 2



benign Image 3



benign Mask 3



```
df_benign = df[df["tumor_type"] == "benign"]
df_malignant = df[df["tumor_type"] == "malignant"]
df_normal = df[df["tumor_type"] == "normal"]

max_size = max(len(df_benign), len(df_malignant), len(df_normal))

from sklearn.utils import resample

df_malignant_oversampled = resample(df_malignant,
                                     replace=True,
                                     n_samples=max_size,
                                     random_state=42)
df_normal_oversampled = resample(df_normal,
                                  replace=True,
                                  n_samples=max_size,
                                  random_state=42)

df_balanced = pd.concat([df_benign, df_malignant_oversampled,
df_normal_oversampled])

df_balanced = df_balanced.sample(frac=1,
random_state=42).reset_index(drop=True)

print("\nBalanced Class Distribution:")
print(df_balanced["tumor_type"].value_counts())
```

```

Balanced Class Distribution:
tumor_type
normal       437
benign       437
malignant    437
Name: count, dtype: int64
```

```
df_balanced
```

```
                                             image_path  \
0     /kaggle/input/breast-ultrasound-images-dataset...
1     /kaggle/input/breast-ultrasound-images-dataset...
2     /kaggle/input/breast-ultrasound-images-dataset...
3     /kaggle/input/breast-ultrasound-images-dataset...
4     /kaggle/input/breast-ultrasound-images-dataset...
...                                                 ...
1306  /kaggle/input/breast-ultrasound-images-dataset...
1307  /kaggle/input/breast-ultrasound-images-dataset...
1308  /kaggle/input/breast-ultrasound-images-dataset...
1309  /kaggle/input/breast-ultrasound-images-dataset...
1310  /kaggle/input/breast-ultrasound-images-dataset...

                                              mask_path tumor_type
0     /kaggle/input/breast-ultrasound-images-dataset...     normal
```

```
1     /kaggle/input/breast-ultrasound-images-dataset...     normal
2     /kaggle/input/breast-ultrasound-images-dataset...     benign
3     /kaggle/input/breast-ultrasound-images-dataset...  malignant
4     /kaggle/input/breast-ultrasound-images-dataset...     benign
...                                                 ...        ...
1306  /kaggle/input/breast-ultrasound-images-dataset...     normal
1307  /kaggle/input/breast-ultrasound-images-dataset...     normal
1308  /kaggle/input/breast-ultrasound-images-dataset...     normal
1309  /kaggle/input/breast-ultrasound-images-dataset...  malignant
1310  /kaggle/input/breast-ultrasound-images-dataset...     normal

[1311 rows x 3 columns]
```

```
!pip install -U albumentations
!pip install segmentation-models-pytorch
```

```
Requirement already satisfied: albumentations in
/usr/local/lib/python3.11/dist-packages (2.0.8)
Requirement already satisfied: numpy>=1.24.4 in
/usr/local/lib/python3.11/dist-packages (from albumentations) (1.26.4)
Requirement already satisfied: scipy>=1.10.0 in
/usr/local/lib/python3.11/dist-packages (from albumentations) (1.15.2)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.11/dist-packages (from albumentations) (6.0.2)
Requirement already satisfied: pydantic>=2.9.2 in
/usr/local/lib/python3.11/dist-packages (from albumentations) (2.11.4)
Requirement already satisfied: albucore==0.0.24 in
/usr/local/lib/python3.11/dist-packages (from albumentations) (0.0.24)
Requirement already satisfied: opencv-python-headless>=4.9.0.80 in
/usr/local/lib/python3.11/dist-packages (from albumentations)
(4.11.0.86)
Requirement already satisfied: stringzilla>=3.10.4 in
/usr/local/lib/python3.11/dist-packages (from albucore==0.0.24-
>albumentations) (3.12.3)
Requirement already satisfied: simsimd>=5.9.2 in
/usr/local/lib/python3.11/dist-packages (from albucore==0.0.24-
>albumentations) (6.2.1)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.4-
>albumentations) (1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.4-
>albumentations) (1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.4-
>albumentations) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-
packages (from numpy>=1.24.4->albumentations) (2025.1.0)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.4-
>albumentations) (2022.1.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.4-
>albumentations) (2.4.1)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.9.2-
>albumentations) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.9.2-
>albumentations) (2.33.2)
Requirement already satisfied: typing-extensions>=4.12.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.9.2-
>albumentations) (4.13.2)
Requirement already satisfied: typing-inspection>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.9.2-
>albumentations) (0.4.0)
Requirement already satisfied: intel-openmp<2026,>=2024 in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.24.4-
>albumentations) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.24.4-
>albumentations) (2022.1.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl-
>numpy>=1.24.4->albumentations) (1.3.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.11/dist-packages (from mkl_umath-
>numpy>=1.24.4->albumentations) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.11/dist-packages (from intel-
openmp<2026,>=2024->mkl->numpy>=1.24.4->albumentations) (2024.2.0)
Requirement already satisfied: segmentation-models-pytorch in
/usr/local/lib/python3.11/dist-packages (0.5.0)
Requirement already satisfied: huggingface-hub>=0.24 in
/usr/local/lib/python3.11/dist-packages (from segmentation-models-
pytorch) (0.31.1)
Requirement already satisfied: numpy>=1.19.3 in
/usr/local/lib/python3.11/dist-packages (from segmentation-models-
pytorch) (1.26.4)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.11/dist-packages (from segmentation-models-
pytorch) (11.1.0)
Requirement already satisfied: safetensors>=0.3.1 in
/usr/local/lib/python3.11/dist-packages (from segmentation-models-
pytorch) (0.5.3)
Requirement already satisfied: timm>=0.9 in
/usr/local/lib/python3.11/dist-packages (from segmentation-models-
pytorch) (1.0.15)
Requirement already satisfied: torch>=1.8 in
```

```
/usr/local/lib/python3.11/dist-packages (from segmentation-models-
pytorch) (2.6.0+cu124)
Requirement already satisfied: torchvision>=0.9 in
/usr/local/lib/python3.11/dist-packages (from segmentation-models-
pytorch) (0.21.0+cu124)
Requirement already satisfied: tqdm>=4.42.1 in
/usr/local/lib/python3.11/dist-packages (from segmentation-models-
pytorch) (4.67.1)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24-
>segmentation-models-pytorch) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24-
>segmentation-models-pytorch) (2025.3.2)
Requirement already satisfied: packaging>=20.9 in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24-
>segmentation-models-pytorch) (25.0)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24-
>segmentation-models-pytorch) (6.0.2)
Requirement already satisfied: requests in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24-
>segmentation-models-pytorch) (2.32.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24-
>segmentation-models-pytorch) (4.13.2)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24-
>segmentation-models-pytorch) (1.1.0)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.19.3-
>segmentation-models-pytorch) (1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.19.3-
>segmentation-models-pytorch) (1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.19.3-
>segmentation-models-pytorch) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-
packages (from numpy>=1.19.3->segmentation-models-pytorch) (2025.1.0)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.19.3-
>segmentation-models-pytorch) (2022.1.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.19.3-
>segmentation-models-pytorch) (2.4.1)
Requirement already satisfied: networkx in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (3.4.2)
```

```
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127
in /usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (12.4.127)
Requirement already satisfied: triton==3.2.0 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8-
>segmentation-models-pytorch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from sympy==1.13.1-
>torch>=1.8->segmentation-models-pytorch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.8-
>segmentation-models-pytorch) (3.0.2)
Requirement already satisfied: intel-openmp<2026,>=2024 in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.19.3-
>segmentation-models-pytorch) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.19.3-
>segmentation-models-pytorch) (2022.1.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl-
>numpy>=1.19.3->segmentation-models-pytorch) (1.3.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.11/dist-packages (from mkl_umath-
>numpy>=1.19.3->segmentation-models-pytorch) (2024.2.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface-
hub>=0.24->segmentation-models-pytorch) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface-
hub>=0.24->segmentation-models-pytorch) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface-
hub>=0.24->segmentation-models-pytorch) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface-
hub>=0.24->segmentation-models-pytorch) (2025.4.26)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.11/dist-packages (from intel-
openmp<2026,>=2024->mkl->numpy>=1.19.3->segmentation-models-pytorch)
(2024.2.0)
```

```python
import pandas as pd
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
import albumentations as A
from albumentations.pytorch import ToTensorV2
import torch
import torch.nn as nn
import segmentation_models_pytorch as smp
from sklearn.metrics import f1_score
from tqdm import tqdm
from torch.optim import Adam
import matplotlib.pyplot as plt
import os
```

```python
class BreastUltrasoundDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform
        self.label_map = {'normal': 0, 'benign': 1, 'malignant': 2}

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_path = self.dataframe.iloc[idx]['image_path']
        mask_path = self.dataframe.iloc[idx]['mask_path']
        label = self.label_map[self.dataframe.iloc[idx]['tumor_type']]
        image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        mask = (mask > 0).astype(np.uint8)
        if self.transform:
            augmented = self.transform(image=image, mask=mask)
            image, mask = augmented['image'],
augmented['mask'].unsqueeze(0)
        return image, mask, label, img_path

train_transform = A.Compose([
    A.Resize(256, 256),
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=30, p=0.5),
    A.Normalize(mean=0.5, std=0.5),
    ToTensorV2()
])

val_transform = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=0.5, std=0.5),
    ToTensorV2()
])

train_df, temp_df = train_test_split(df_balanced, test_size=0.3,
stratify=df_balanced['tumor_type'], random_state=42)
val_df, test_df = train_test_split(temp_df, test_size=0.5,
stratify=temp_df['tumor_type'], random_state=42)

train_dataset = BreastUltrasoundDataset(train_df,
transform=train_transform)
val_dataset = BreastUltrasoundDataset(val_df, transform=val_transform)
test_dataset = BreastUltrasoundDataset(test_df,
transform=val_transform)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

```
class MultiTaskModel(nn.Module):
    def __init__(self, backbone='resnet34', num_classes=3):
        super(MultiTaskModel, self).__init__()
        self.backbone = smp.UnetPlusPlus(
            encoder_name=backbone,
            encoder_weights='imagenet',
            in_channels=1,
            classes=1,
            activation='sigmoid'
        )
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(512, num_classes)

    def forward(self, x):
        seg_output = self.backbone(x)
        enc_features = self.backbone.encoder(x)[-1]
        cls_output =
self.pool(enc_features).view(enc_features.size(0), -1)
        cls_output = self.fc(cls_output)
        normal_mask = (torch.argmax(cls_output, dim=1) ==
0).float().unsqueeze(1).unsqueeze(2).unsqueeze(3)
        normal_mask = normal_mask.expand(-1, -1, seg_output.size(2),
seg_output.size(3))
        seg_output = seg_output * (1 - normal_mask)
        return seg_output, cls_output

class DiceLoss(nn.Module):
    def __init__(self):
        super(DiceLoss, self).__init__()

    def forward(self, pred, target, smooth=1):
        pred = pred.contiguous().view(-1)
        target = target.contiguous().view(-1)
        intersection = (pred * target).sum()
        return 1 - ((2. * intersection + smooth) / (pred.sum() +
target.sum() + smooth))

dice_loss = DiceLoss()
bce_loss = nn.BCEWithLogitsLoss()
ce_loss = nn.CrossEntropyLoss()

def combined_loss(seg_pred, seg_target, cls_pred, cls_target,
w_seg=0.5, w_cls=0.5):
    seg_loss = 0.5 * dice_loss(seg_pred, seg_target) + 0.5 *
bce_loss(seg_pred, seg_target)
    cls_loss = ce_loss(cls_pred, cls_target)
    return w_seg * seg_loss + w_cls * cls_loss

def refine_predictions(seg_pred, cls_pred):
    normal_probs = torch.softmax(cls_pred, dim=1)[:,
0].unsqueeze(1).unsqueeze(2).unsqueeze(3)
    normal_probs = normal_probs.expand(-1, -1, seg_pred.size(2),
seg_pred.size(3))
    seg_pred = seg_pred * (1 - normal_probs)
    return seg_pred

model = MultiTaskModel(backbone='resnet34', num_classes=3).cuda()
optimizer = Adam(model.parameters(), lr=1e-3)
num_epochs = 5

for epoch in range(num_epochs):
    model.train()
    train_loss = 0
    for images, masks, labels, _ in tqdm(train_loader):
        images, masks, labels = images.cuda(), masks.cuda(),
labels.cuda()
        optimizer.zero_grad()
        seg_pred, cls_pred = model(images)
        seg_pred = refine_predictions(seg_pred, cls_pred)
        loss = combined_loss(seg_pred, masks.float(), cls_pred,
labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    print(f'Epoch {epoch+1}, Train Loss: {train_loss /
len(train_loader)}')

    model.eval()
    val_loss = 0
    with torch.no_grad():
        for images, masks, labels, _ in val_loader:
            images, masks, labels = images.cuda(), masks.cuda(),
labels.cuda()
            seg_pred, cls_pred = model(images)
            seg_pred = refine_predictions(seg_pred, cls_pred)
            loss = combined_loss(seg_pred, masks.float(), cls_pred,
labels)
            val_loss += loss.item()
    print(f'Epoch {epoch+1}, Val Loss: {val_loss / len(val_loader)}')

model_path = '/kaggle/working/model.pth'
torch.save(model.state_dict(), model_path)
print(f"Model weights saved to {model_path}.")

def compute_dsc(pred, target, smooth=1):
    pred = (pred > 0.5).float()
    intersection = (pred * target).sum()
    return (2. * intersection + smooth) / (pred.sum() + target.sum() +
smooth)
```

```
save_dir = '/kaggle/working/predicted_masks'
os.makedirs(save_dir, exist_ok=True)

sample_images = []
sample_gt_masks = []
sample_pred_masks = []
sample_labels = []
sample_paths = []

model.eval()
dsc_scores = {'benign': [], 'malignant': [], 'normal': []}
f1_scores = {'benign': [], 'malignant': [], 'normal': []}
all_preds, all_labels = [], []

with torch.no_grad():
    for images, masks, labels, img_paths in test_loader:
        images, masks, labels = images.cuda(), masks.cuda(),
labels.cuda()
        seg_pred, cls_pred = model(images)
        seg_pred = refine_predictions(seg_pred, cls_pred)

        seg_pred_np = (seg_pred > 0.5).float().cpu().numpy()
        for i, (pred, path) in enumerate(zip(seg_pred_np, img_paths)):
            pred_mask = (pred[0] * 255).astype(np.uint8)
            filename = os.path.basename(path).replace('.png',
'_pred_mask.png')
            cv2.imwrite(os.path.join(save_dir, filename), pred_mask)

        if len(sample_images) < 10:
            sample_images.extend(images.cpu().numpy()[:min(10-
len(sample_images), len(images))])
            sample_gt_masks.extend(masks.cpu().numpy()[:min(10-
len(sample_gt_masks), len(masks))])
            sample_pred_masks.extend(seg_pred_np[:min(10-
len(sample_pred_masks), len(seg_pred_np))])
            sample_labels.extend(labels.cpu().numpy()[:min(10-
len(sample_labels), len(labels))])
            sample_paths.extend(img_paths[:min(10-len(sample_paths),
len(img_paths))])

        for i, label in enumerate(labels):
            tumor_type = {0: 'normal', 1: 'benign', 2: 'malignant'}
[label.item()]
            dsc = compute_dsc(seg_pred[i], masks[i]).item()
            dsc_scores[tumor_type].append(dsc)

        preds = torch.argmax(cls_pred, dim=1).cpu().numpy()
        labels_np = labels.cpu().numpy()
        all_preds.extend(preds)
        all_labels.extend(labels_np)

for tumor_type in dsc_scores:
    print(f'DSC {tumor_type}: {np.mean(dsc_scores[tumor_type]):.3f} ±
{np.std(dsc_scores[tumor_type]):.3f}')

f1_per_class = f1_score(all_labels, all_preds, average=None)
f1_weighted = f1_score(all_labels, all_preds, average='weighted')
print(f'F1 Benign: {f1_per_class[1]:.3f}, F1 Malignant:
{f1_per_class[2]:.3f}, F1 Normal: {f1_per_class[0]:.3f}')
print(f'F1 Weighted: {f1_weighted:.3f}')

fig, axes = plt.subplots(10, 3, figsize=(15, 30))
label_map = {0: 'Normal', 1: 'Benign', 2: 'Malignant'}
for i in range(min(10, len(sample_images))):
    img = sample_images[i][0] * 0.5 + 0.5
    gt_mask = sample_gt_masks[i][0]
    pred_mask = sample_pred_masks[i][0]

    axes[i, 0].imshow(img, cmap='gray')
    axes[i, 0].set_title(f'Image ({label_map[sample_labels[i]]})')
    axes[i, 0].axis('off')

    axes[i, 1].imshow(gt_mask, cmap='gray')
    axes[i, 1].set_title('Ground Truth Mask')
    axes[i, 1].axis('off')

    axes[i, 2].imshow(pred_mask, cmap='gray')
    axes[i, 2].set_title('Predicted Mask')
    axes[i, 2].axis('off')

plt.tight_layout()
plt.savefig('/kaggle/working/visualization.png')
plt.show()

def display_saved_mask(image_path,
save_dir='/kaggle/working/predicted_masks'):
    filename = os.path.basename(image_path).replace('.png',
'_pred_mask.png')
    pred_mask_path = os.path.join(save_dir, filename)
    if os.path.exists(pred_mask_path):
        pred_mask = cv2.imread(pred_mask_path, cv2.IMREAD_GRAYSCALE)
        plt.figure(figsize=(5, 5))
        plt.imshow(pred_mask, cmap='gray')
        plt.title(f'Predicted Mask for
{os.path.basename(image_path)}')
        plt.axis('off')

plt.savefig(f'/kaggle/working/pred_mask_{os.path.basename(image_path)}
.png')  # Save individual mask plot
```

```
        plt.show()
    else:
        print(f'Predicted mask not found: {pred_mask_path}')

if sample_paths:
    for path in sample_paths[:3]:
        display_saved_mask(path)
```

```
100%|██████████| 58/58 [00:46<00:00,  1.25it/s]

Epoch 1, Train Loss: 0.8341118662521757
Epoch 1, Val Loss: 0.9949450492858887

100%|██████████| 58/58 [00:45<00:00,  1.27it/s]

Epoch 2, Train Loss: 0.6296808704220015
Epoch 2, Val Loss: 0.6374650322473966

100%|██████████| 58/58 [00:45<00:00,  1.27it/s]

Epoch 3, Train Loss: 0.51266820266329
Epoch 3, Val Loss: 0.4362595104254209

100%|██████████| 58/58 [00:45<00:00,  1.27it/s]

Epoch 4, Train Loss: 0.511211470283311
Epoch 4, Val Loss: 0.4953395380423619

100%|██████████| 58/58 [00:46<00:00,  1.26it/s]

Epoch 5, Train Loss: 0.4810506074592985
Epoch 5, Val Loss: 0.45303332576384914
Model weights saved to /kaggle/working/model.pth.
DSC benign: 0.633 ± 0.321
DSC malignant: 0.637 ± 0.254
DSC normal: 0.940 ± 0.238
F1 Benign: 0.781, F1 Malignant: 0.768, F1 Normal: 0.931
F1 Weighted: 0.827
```

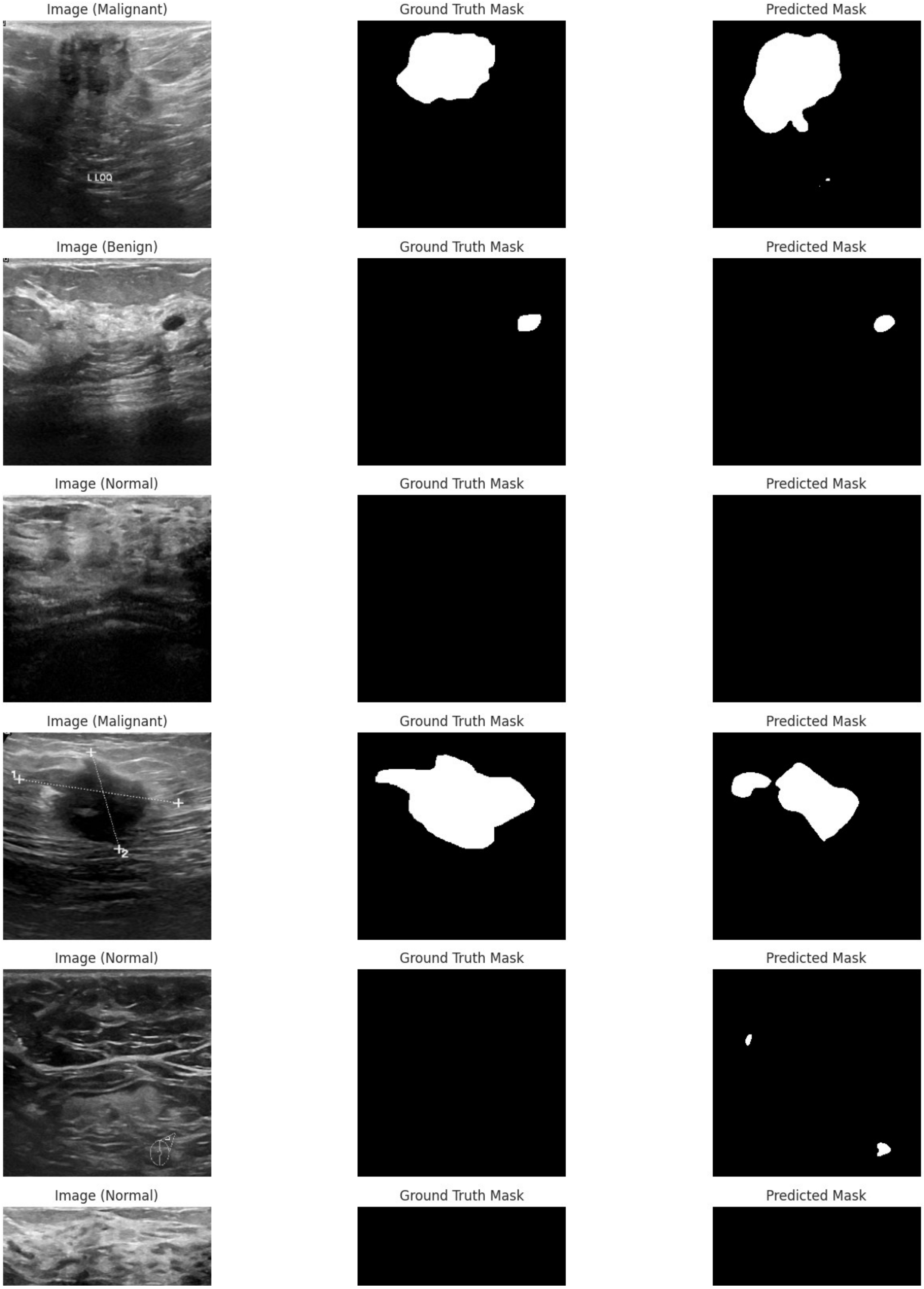
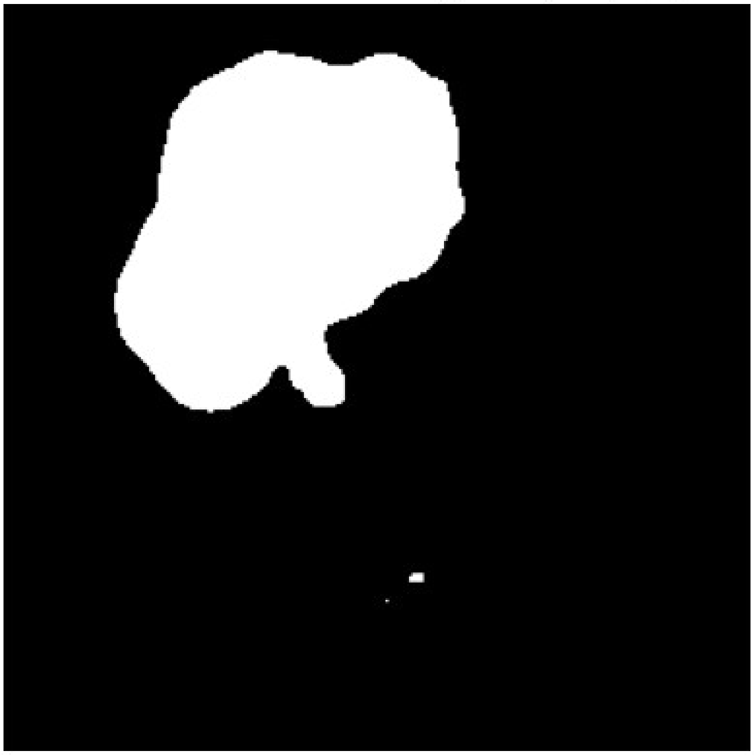
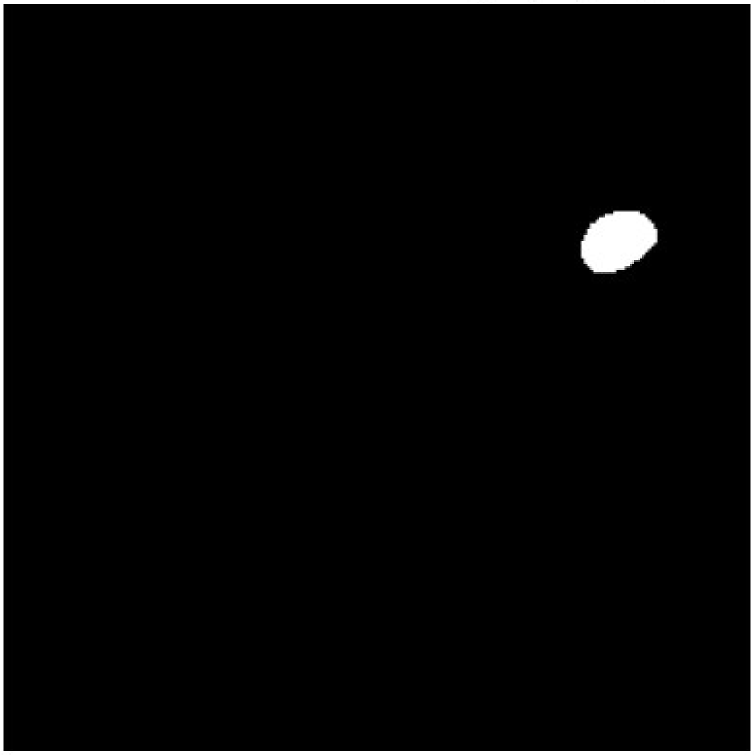


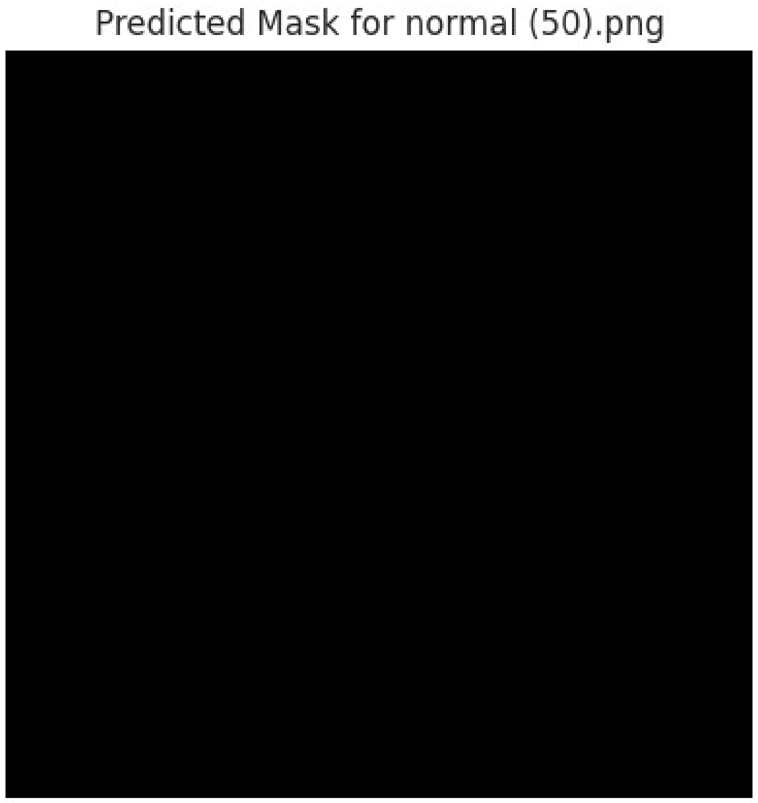
Image (Malignant)
Ground Truth Mask
Predicted Mask
L LOQ
Image (Benign)
Ground Truth Mask
Predicted Mask
Image (Normal)
Ground Truth Mask
Predicted Mask
Image (Malignant)
Ground Truth Mask
Predicted Mask
1
2
Image (Normal)
Ground Truth Mask
Predicted Mask
Image (Normal)
Ground Truth Mask
Predicted Mask



Predicted Mask for malignant (139).png



Predicted Mask for benign (74).png



Predicted Mask for normal (50).png

```
import pandas as pd
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
import albumentations as A
from albumentations.pytorch import ToTensorV2
import torch
import torch.nn as nn
import segmentation_models_pytorch as smp
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
import os

class BreastUltrasoundDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform
        self.label_map = {'normal': 0, 'benign': 1, 'malignant': 2}

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_path = self.dataframe.iloc[idx]['image_path']
        mask_path = self.dataframe.iloc[idx]['mask_path']
        label = self.label_map[self.dataframe.iloc[idx]['tumor_type']]
        image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        mask = (mask > 0).astype(np.uint8)
        if self.transform:
            augmented = self.transform(image=image, mask=mask)
            image, mask = augmented['image'],
augmented['mask'].unsqueeze(0)
        return image, mask, label, img_path

val_transform = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=0.5, std=0.5),
    ToTensorV2()
])

train_df, temp_df = train_test_split(df_balanced, test_size=0.3,
stratify=df_balanced['tumor_type'], random_state=42)
val_df, test_df = train_test_split(temp_df, test_size=0.5,
stratify=temp_df['tumor_type'], random_state=42)

test_dataset = BreastUltrasoundDataset(test_df,
transform=val_transform)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

class MultiTaskModel(nn.Module):
    def __init__(self, backbone='resnet34', num_classes=3):
        super(MultiTaskModel, self).__init__()
        self.backbone = smp.UnetPlusPlus(
            encoder_name=backbone,
            encoder_weights='imagenet',
            in_channels=1,
            classes=1,
            activation='sigmoid'
        )
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(512, num_classes)

    def forward(self, x):
        seg_output = self.backbone(x)
        enc_features = self.backbone.encoder(x)[-1]
        cls_output =
self.pool(enc_features).view(enc_features.size(0), -1)
        cls_output = self.fc(cls_output)
        normal_mask = (torch.argmax(cls_output, dim=1) ==
0).float().unsqueeze(1).unsqueeze(2).unsqueeze(3)
        normal_mask = normal_mask.expand(-1, -1, seg_output.size(2),
seg_output.size(3))
        seg_output = seg_output * (1 - normal_mask)
        return seg_output, cls_output

def refine_predictions(seg_pred, cls_pred):
    normal_probs = torch.softmax(cls_pred, dim=1)[:, 0].unsqueeze(1).unsqueeze(2).unsqueeze(3)
    normal_probs = normal_probs.expand(-1, -1, seg_pred.size(2), seg_pred.size(3))
    seg_pred = seg_pred * (1 - normal_probs)
    return seg_pred

model = MultiTaskModel(backbone='resnet34', num_classes=3).cuda()

model_path = '/kaggle/working/model.pth'
try:
    model.load_state_dict(torch.load(model_path))
    print(f"Loaded trained model weights from {model_path}.")
except FileNotFoundError:
    print(f"Model weights not found at {model_path}. Please ensure the model was trained and saved.")
    exit()

model.eval()

def compute_dsc(pred, target, smooth=1):
    pred = (pred > 0.5).float()
    intersection = (pred * target).sum()
    return (2. * intersection + smooth) / (pred.sum() + target.sum() + smooth)

save_dir = '/kaggle/working/predicted_masks'
os.makedirs(save_dir, exist_ok=True)

sample_images = []
sample_gt_masks = []
sample_pred_masks = []
sample_labels = []
sample_paths = []

dsc_scores = {'benign': [], 'malignant': [], 'normal': []}
f1_scores = {'benign': [], 'malignant': [], 'normal': []}
all_preds, all_labels = [], []

with torch.no_grad():
    for images, masks, labels, img_paths in test_loader:
        images, masks, labels = images.cuda(), masks.cuda(), labels.cuda()
        seg_pred, cls_pred = model(images)
        seg_pred = refine_predictions(seg_pred, cls_pred)

        seg_pred_np = (seg_pred > 0.5).float().cpu().numpy()
        for i, (pred, path) in enumerate(zip(seg_pred_np, img_paths)):
            pred_mask = (pred[0] * 255).astype(np.uint8)
            filename = os.path.basename(path).replace('.png',
'_pred_mask.png')
            cv2.imwrite(os.path.join(save_dir, filename), pred_mask)

        if len(sample_images) < 10:
            sample_images.extend(images.cpu().numpy()[:min(10-
len(sample_images), len(images))])
            sample_gt_masks.extend(masks.cpu().numpy()[:min(10-
len(sample_gt_masks), len(masks))])
            sample_pred_masks.extend(seg_pred_np[:min(10-
len(sample_pred_masks), len(seg_pred_np))])
            sample_labels.extend(labels.cpu().numpy()[:min(10-
len(sample_labels), len(labels))])
            sample_paths.extend(img_paths[:min(10-len(sample_paths),
len(img_paths))])

        for i, label in enumerate(labels):
            tumor_type = {0: 'normal', 1: 'benign', 2: 'malignant'}
[label.item()]
            dsc = compute_dsc(seg_pred[i], masks[i]).item()
            dsc_scores[tumor_type].append(dsc)

        preds = torch.argmax(cls_pred, dim=1).cpu().numpy()
        labels_np = labels.cpu().numpy()
        all_preds.extend(preds)
        all_labels.extend(labels_np)

for tumor_type in dsc_scores:
    print(f'DSC {tumor_type}: {np.mean(dsc_scores[tumor_type]):.3f} ±
{np.std(dsc_scores[tumor_type]):.3f}')

f1_per_class = f1_score(all_labels, all_preds, average=None)
f1_weighted = f1_score(all_labels, all_preds, average='weighted')
print(f'F1 Benign: {f1_per_class[1]:.3f}, F1 Malignant:
{f1_per_class[2]:.3f}, F1 Normal: {f1_per_class[0]:.3f}')
print(f'F1 Weighted: {f1_weighted:.3f}')

fig, axes = plt.subplots(10, 3, figsize=(15, 30))
label_map = {0: 'Normal', 1: 'Benign', 2: 'Malignant'}
for i in range(min(10, len(sample_images))):
    img = sample_images[i][0] * 0.5 + 0.5
    gt_mask = sample_gt_masks[i][0]
    pred_mask = sample_pred_masks[i][0]

    axes[i, 0].imshow(img, cmap='gray')
    axes[i, 0].set_title(f'Image ({label_map[sample_labels[i]]})')
    axes[i, 0].axis('off')
    axes[i, 1].imshow(gt_mask, cmap='gray')
    axes[i, 1].set_title('Ground Truth Mask')
    axes[i, 1].axis('off')

    axes[i, 2].imshow(pred_mask, cmap='gray')
    axes[i, 2].set_title('Predicted Mask')
    axes[i, 2].axis('off')

plt.tight_layout()
plt.savefig('/kaggle/working/visualization.png')
plt.show()

def display_saved_mask(image_path,
save_dir='/kaggle/working/predicted_masks'):
    filename = os.path.basename(image_path).replace('.png',
'_pred_mask.png')
    pred_mask_path = os.path.join(save_dir, filename)
    if os.path.exists(pred_mask_path):
        pred_mask = cv2.imread(pred_mask_path, cv2.IMREAD_GRAYSCALE)
        plt.figure(figsize=(5, 5))
        plt.imshow(pred_mask, cmap='gray')
        plt.title(f'Predicted Mask for
{os.path.basename(image_path)}')
        plt.axis('off')

plt.savefig(f'/kaggle/working/pred_mask_{os.path.basename(image_path)}
.png')
        plt.show()
    else:
        print(f'Predicted mask not found: {pred_mask_path}')

if sample_paths:
    for path in sample_paths[:3]:
        display_saved_mask(path)
```

```
Loaded trained model weights from /kaggle/working/model.pth.
DSC benign: 0.633 ± 0.321
DSC malignant: 0.637 ± 0.254
DSC normal: 0.940 ± 0.238
F1 Benign: 0.781, F1 Malignant: 0.768, F1 Normal: 0.931
F1 Weighted: 0.827
```

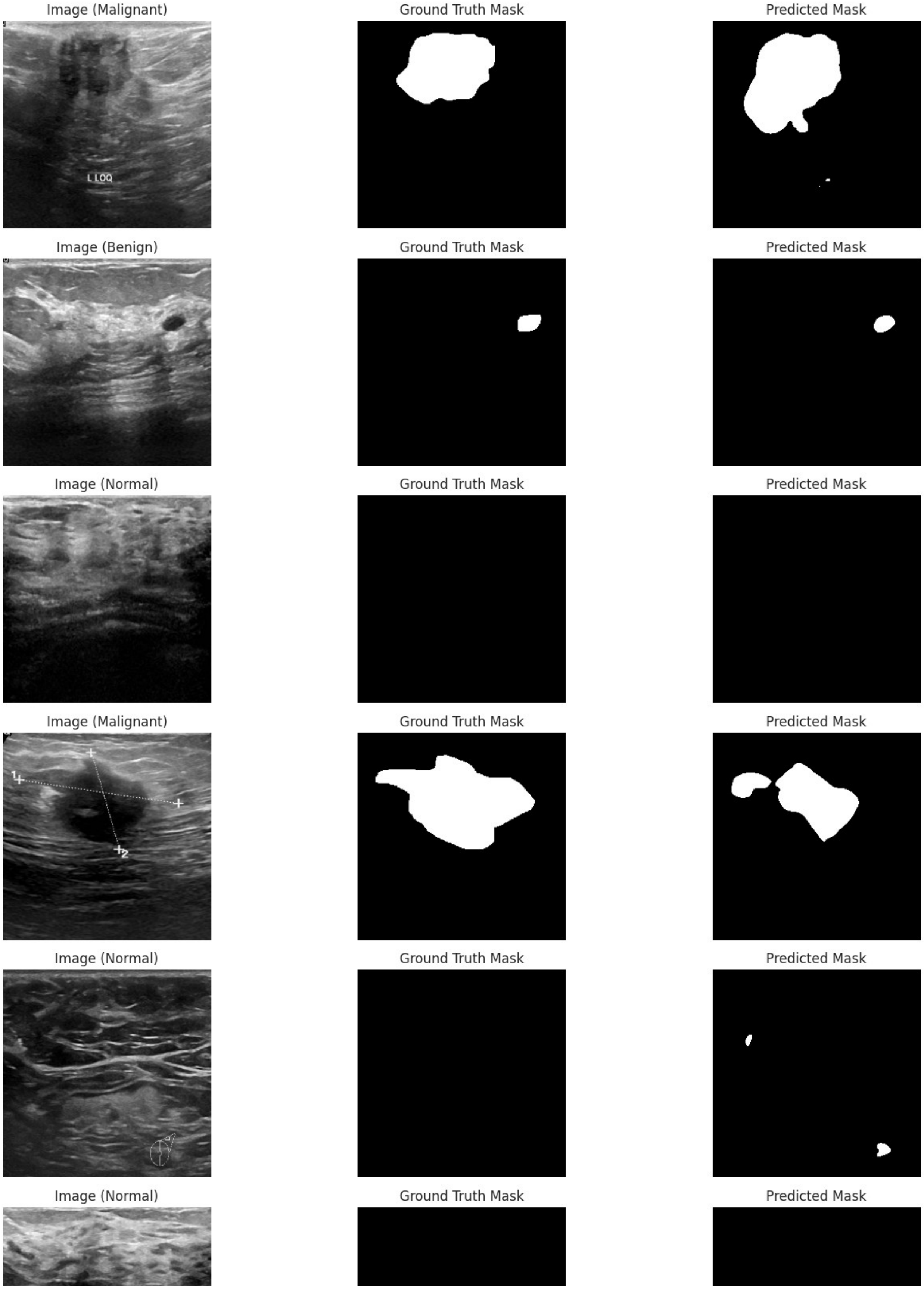
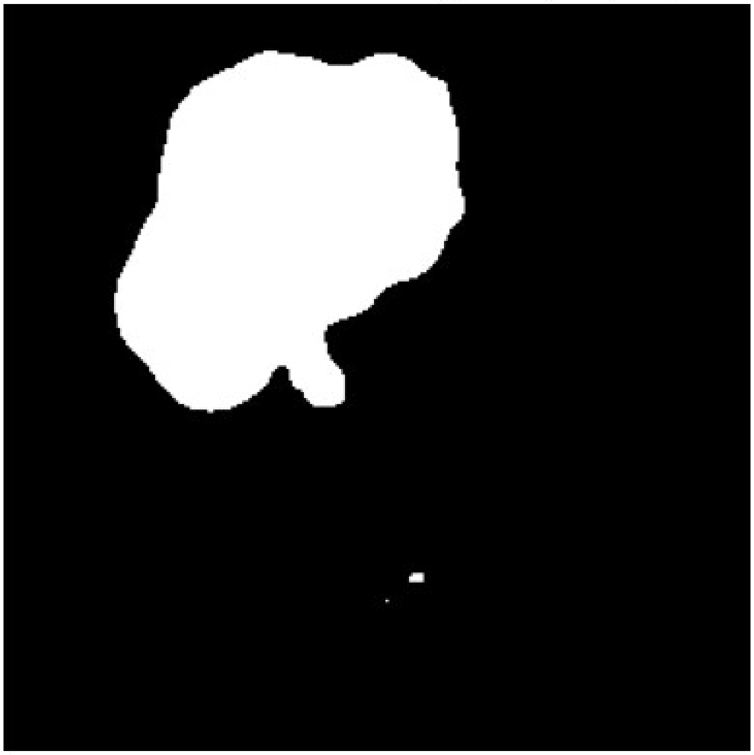
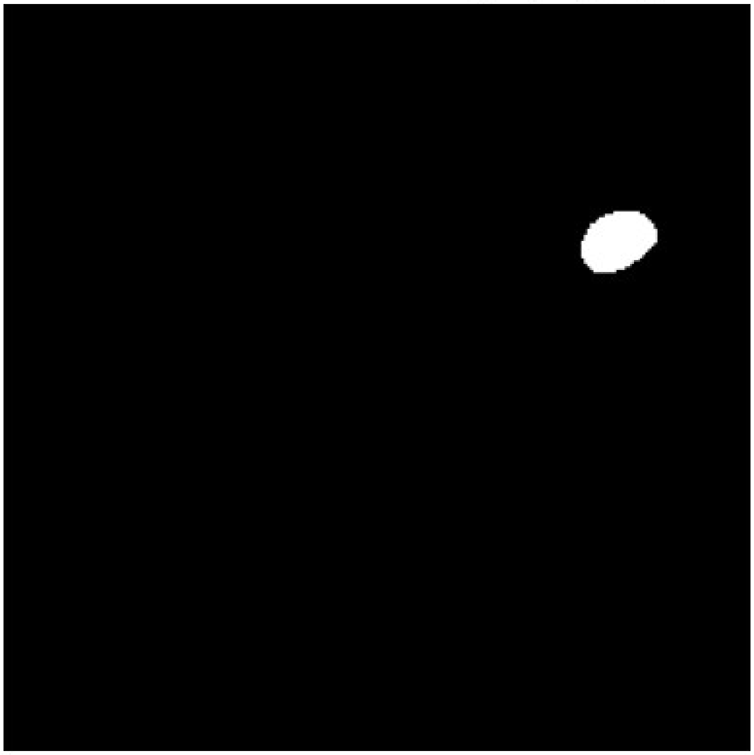


Image (Malignant)
Ground Truth Mask
Predicted Mask
L LOQ
Image (Benign)
Ground Truth Mask
Predicted Mask
Image (Normal)
Ground Truth Mask
Predicted Mask
Image (Malignant)
Ground Truth Mask
Predicted Mask
1
2
Image (Normal)
Ground Truth Mask
Predicted Mask
Image (Normal)
Ground Truth Mask
Predicted Mask



Predicted Mask for malignant (139).png



Predicted Mask for benign (74).png

Predicted Mask for normal (50).png