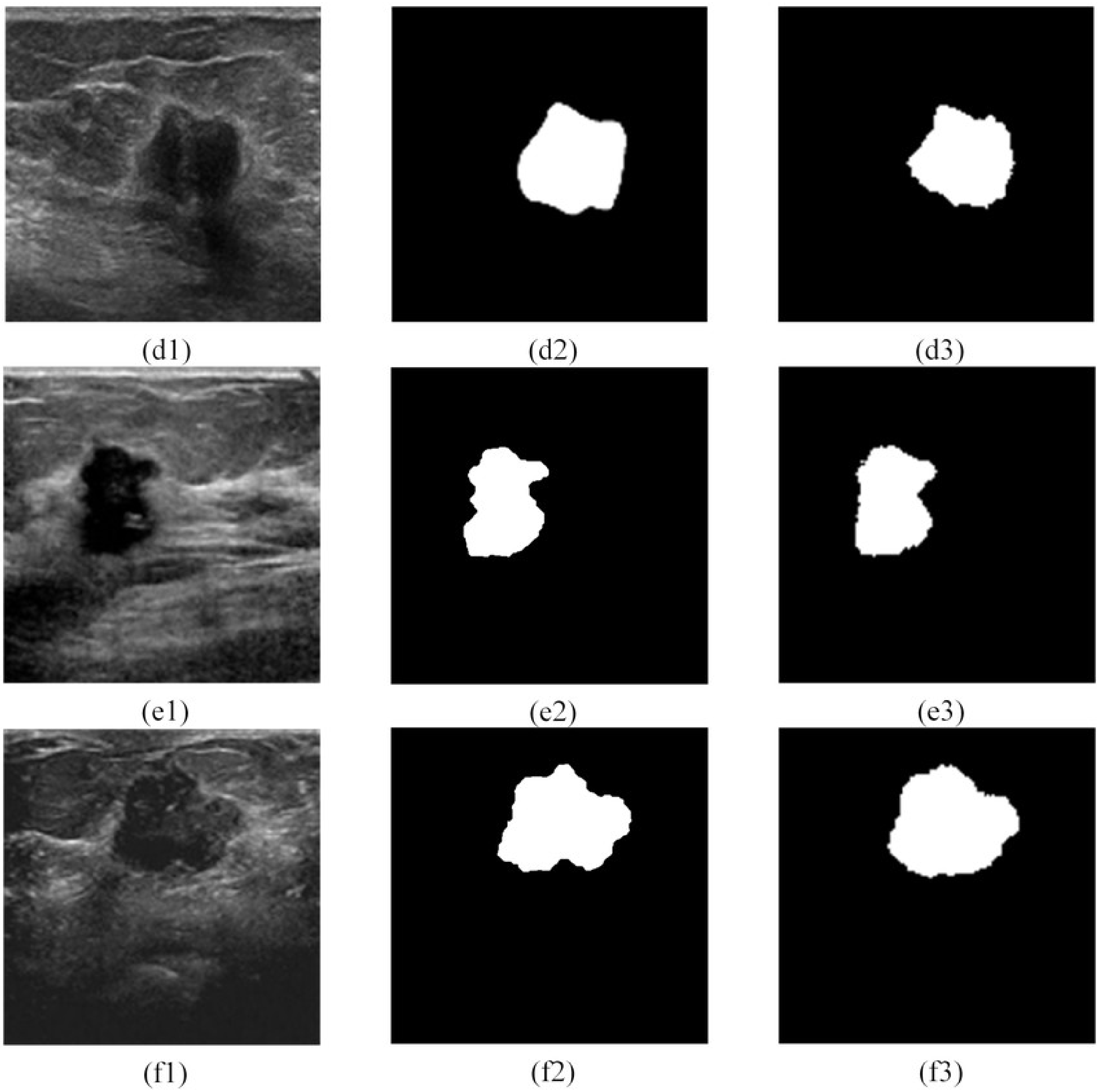
Breast Cancer Segmentation using Hybrid Attention

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



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

```python
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
```

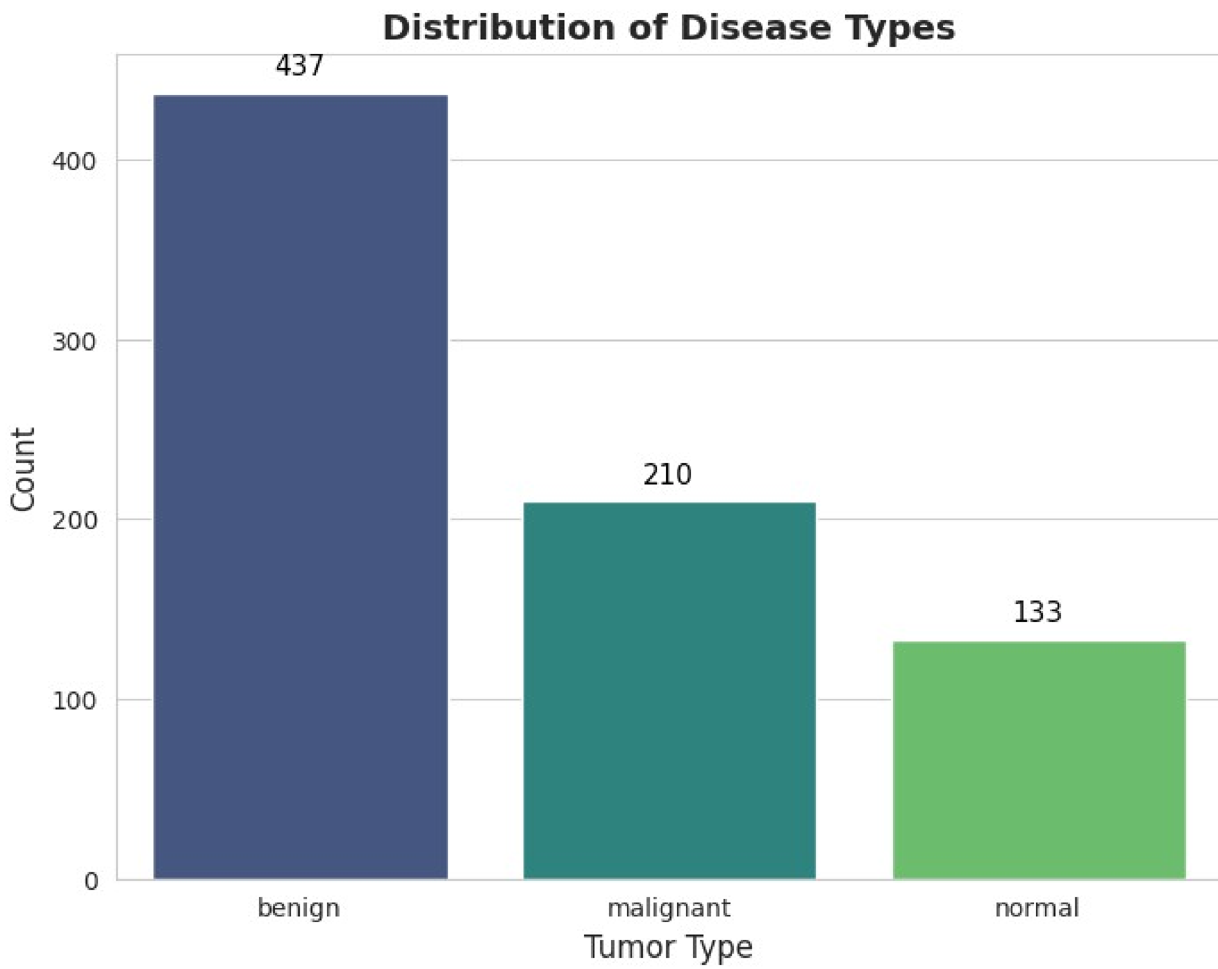
```
df['tumor_type'].unique()
```

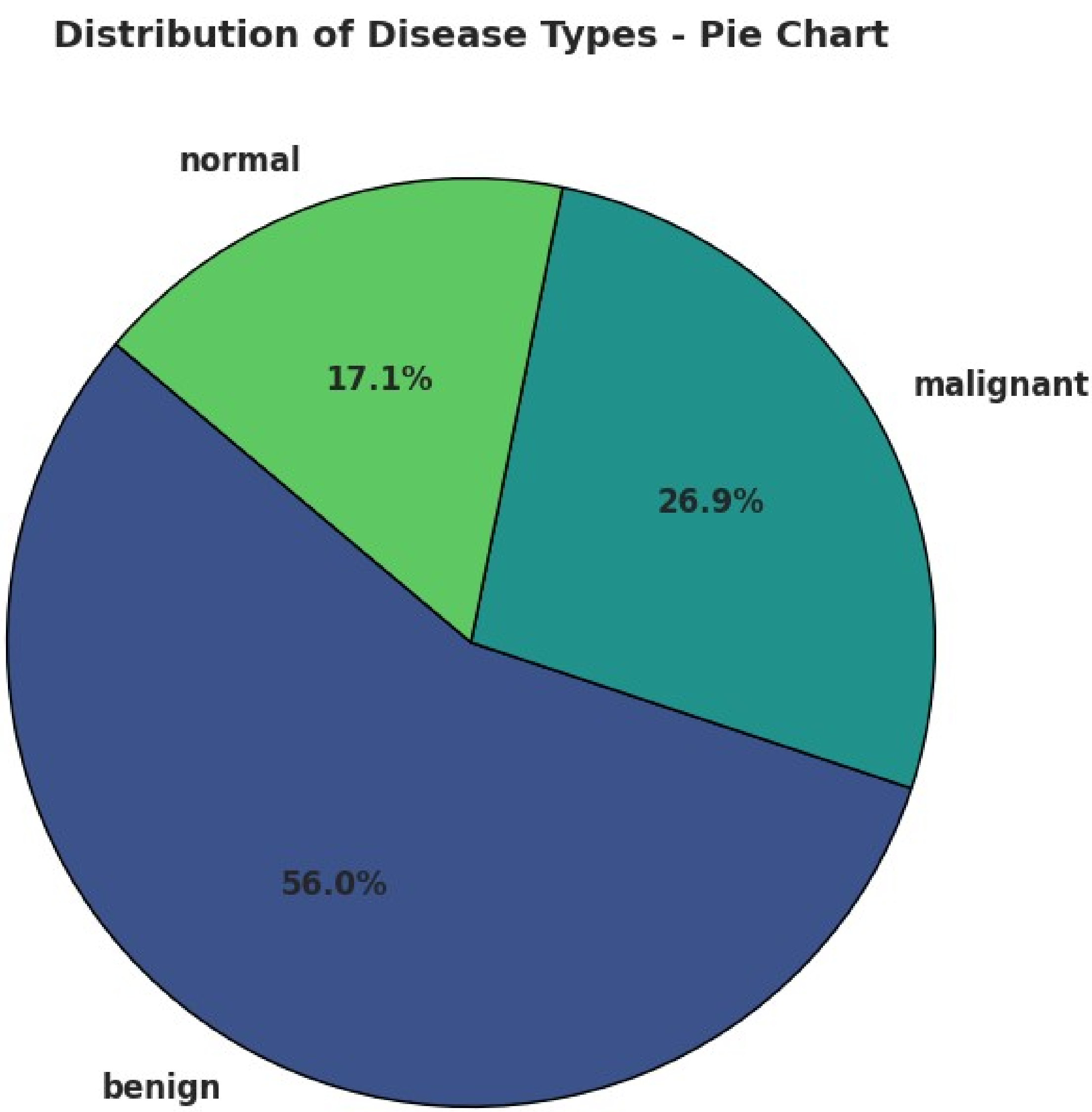
```
array(['benign', 'malignant', 'normal'], dtype=object)
```

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

```
tumor_type
benign       437
malignant    210
normal       133
Name: count, dtype: int64
```

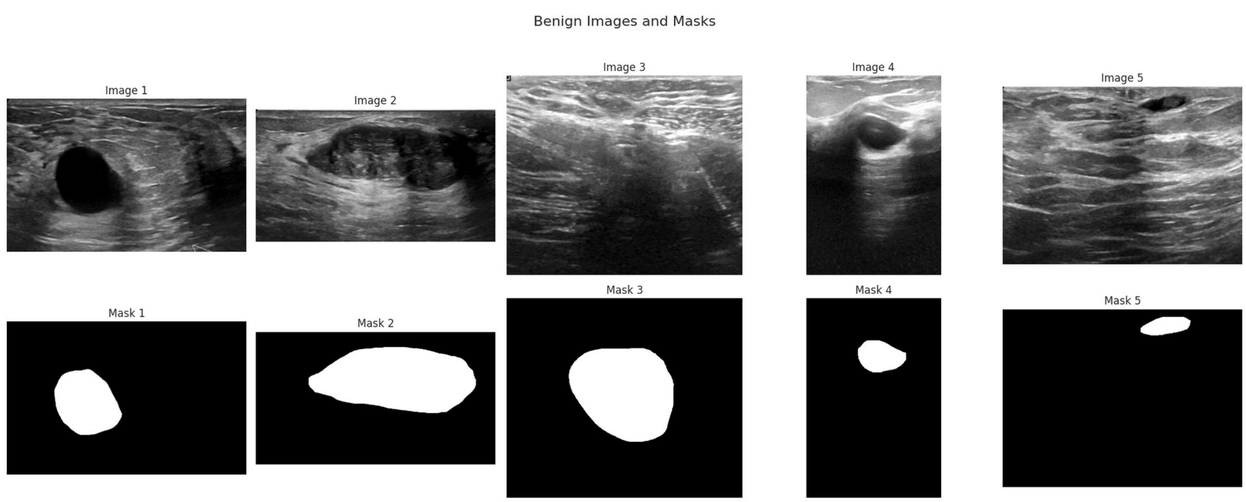
```python
import seaborn as sns
import matplotlib.pyplot as plt

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()
```

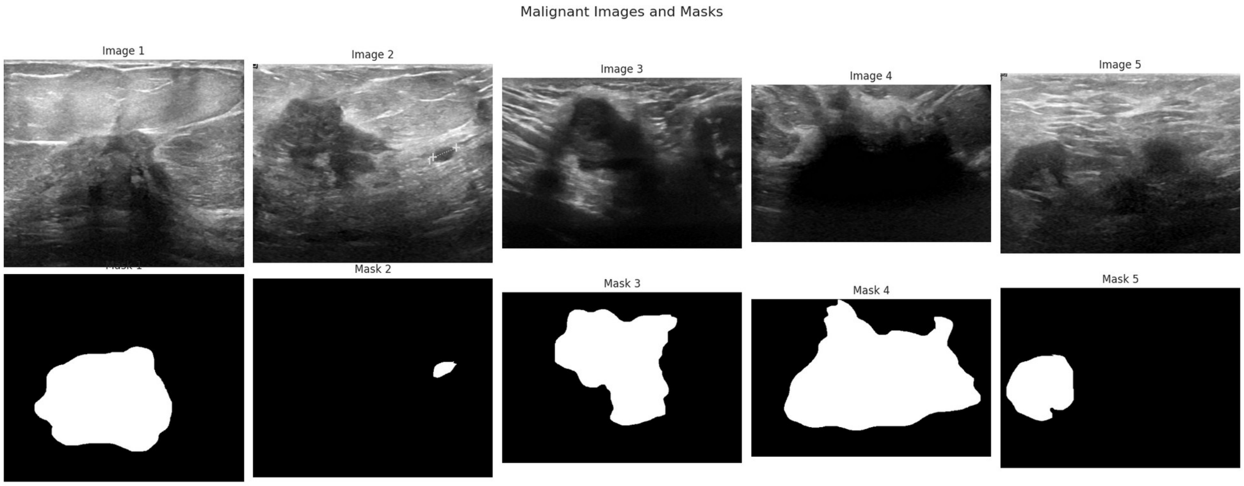

Distribution of Disease Types
437
210
133
400
300
200
100
0
Count
benign
malignant
normal
Tumor Type

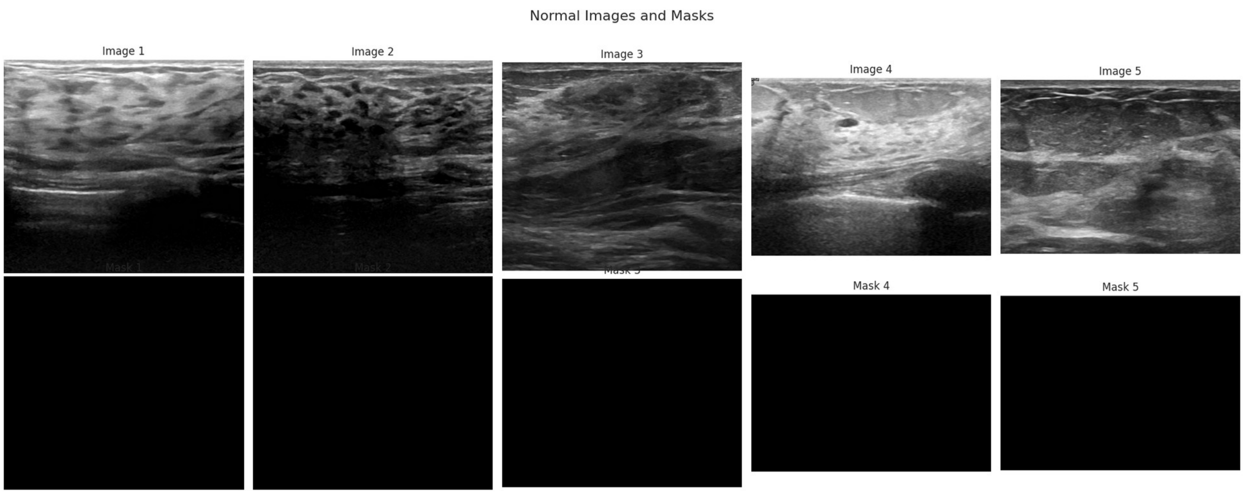



```
from PIL import Image

def display_images_and_masks(df, tumor_type, num_samples=5):

    category_df = df[df['tumor_type'] ==
tumor_type].sample(n=num_samples, random_state=42)


    fig, axes = plt.subplots(2, num_samples, figsize=(num_samples * 4,
8))
    fig.suptitle(f'{tumor_type.capitalize()} Images and Masks',
fontsize=16)
```

```
    for idx, (_, row) in enumerate(category_df.iterrows()):
        try:

            image = Image.open(row['image_path'])
            mask = Image.open(row['mask_path'])

            image = np.array(image)
            mask = np.array(mask)

            axes[0, idx].imshow(image, cmap='gray')
            axes[0, idx].set_title(f'Image {idx+1}')
            axes[0, idx].axis('off')

            axes[1, idx].imshow(mask, cmap='gray')
            axes[1, idx].set_title(f'Mask {idx+1}')
            axes[1, idx].axis('off')

        except FileNotFoundError:
            print(f"Error: Could not load image or mask at index {idx} for {tumor_type}")
            axes[0, idx].axis('off')
            axes[1, idx].axis('off')

    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()

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

for tumor_type in tumor_types:
    display_images_and_masks(df, tumor_type, num_samples=5)
```








```
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]
```

```
import tensorflow as tf
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.layers import Layer, Input, Conv2D,
UpSampling2D, Concatenate, BatchNormalization, ReLU,
GlobalAveragePooling2D, GlobalMaxPooling2D, Dense, Reshape, Multiply,
Add
from tensorflow.keras.models import Model
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping

# Suppress CUDA warnings
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

# Define custom loss function: BCE + Jaccard (IoU)
def hybrid_loss(y_true, y_pred):
    bce = tf.keras.losses.BinaryCrossentropy()(y_true, y_pred)
    intersection = tf.reduce_sum(y_true * y_pred, axis=[1, 2, 3])
    union = tf.reduce_sum(y_true + y_pred, axis=[1, 2, 3]) -
intersection
    iou = intersection / (union + tf.keras.backend.epsilon())
    iou_loss = -tf.reduce_mean(tf.math.log(iou +
tf.keras.backend.epsilon()))
    return bce + iou_loss

# Spatial Feature Enhancement Block (SFEB)
def SFEB(x, filters):
    input_channels = x.shape[-1]
    if input_channels != filters:
        x = Conv2D(filters, 1, padding='same')(x)
    conv = Conv2D(filters, 3, padding='same')(x)
    conv = BatchNormalization()(conv)
    conv = ReLU()(conv)
    gmp = GlobalMaxPooling2D()(conv)
    gap = GlobalAveragePooling2D()(conv)
    pooled = Concatenate()([gmp, gap])
    dense = Dense(filters, activation='relu')(pooled)
    dense = Dense(filters, activation='sigmoid')(dense)
    attention = Reshape((1, 1, filters))(dense)
    attention = Multiply()([conv, attention])
    out = Add()([x, attention])
    return out

# Custom Keras Layer for Transformer Self-Attention (TSA)
class TSALayer(Layer):
    def __init__(self, filters, **kwargs):
        super(TSALayer, self).__init__(**kwargs)
        self.filters = filters
        self.pos_dense = Dense(filters, activation='relu')
        self.q_conv = Conv2D(filters, 1)
        self.k_conv = Conv2D(filters, 1)
        self.v_conv = Conv2D(filters, 1)
        self.bn = BatchNormalization()

    def call(self, x):
        pos_encoding = self.pos_dense(x)
```

```
        x = Add()([x, pos_encoding])
        q = self.q_conv(x)
        k = self.k_conv(x)
        v = self.v_conv(x)
        batch_size = tf.shape(q)[0]
        height = tf.shape(q)[1]
        width = tf.shape(q)[2]
        q = tf.reshape(q, [batch_size, height * width, self.filters])
        k = tf.reshape(k, [batch_size, height * width, self.filters])
        v = tf.reshape(v, [batch_size, height * width, self.filters])
        attention_scores = tf.matmul(q, k, transpose_b=True)
        attention_scores = attention_scores /
tf.sqrt(tf.cast(self.filters, tf.float32))
        attention_weights = tf.nn.softmax(attention_scores, axis=-1)
        attention_out = tf.matmul(attention_weights, v)
        attention_out = tf.reshape(attention_out, [batch_size, height,
width, self.filters])
        out = Add()([x, attention_out])
        out = self.bn(out)
        return out

# Custom Keras Layer for Global Spatial Attention (GSA)
class GSALayer(Layer):
    def __init__(self, filters, **kwargs):
        super(GSALayer, self).__init__(**kwargs)
        self.filters = filters
        self.f1_conv = Conv2D(filters // 2, 1)
        self.f2_conv = Conv2D(filters // 2, 1)
        self.v_conv = Conv2D(filters, 1)
        self.out_conv = Conv2D(filters, 1, activation='relu')

    def call(self, x):
        c = self.filters // 2
        f1 = self.f1_conv(x)
        f2 = self.f2_conv(x)
        v = self.v_conv(x)
        batch_size = tf.shape(f1)[0]
        height = tf.shape(f1)[1]
        width = tf.shape(f1)[2]
        f1 = tf.reshape(f1, [batch_size, height * width, c])
        f2 = tf.reshape(f2, [batch_size, height * width, c])
        v = tf.reshape(v, [batch_size, height * width, self.filters])
        attention_scores = tf.matmul(f1, f2, transpose_b=True)
        attention_weights = tf.nn.softmax(attention_scores, axis=-1)
        out = tf.matmul(attention_weights, v)
        out = tf.reshape(out, [batch_size, height, width,
self.filters])
        out = Concatenate()([x, out])
        out = self.out_conv(out)
```

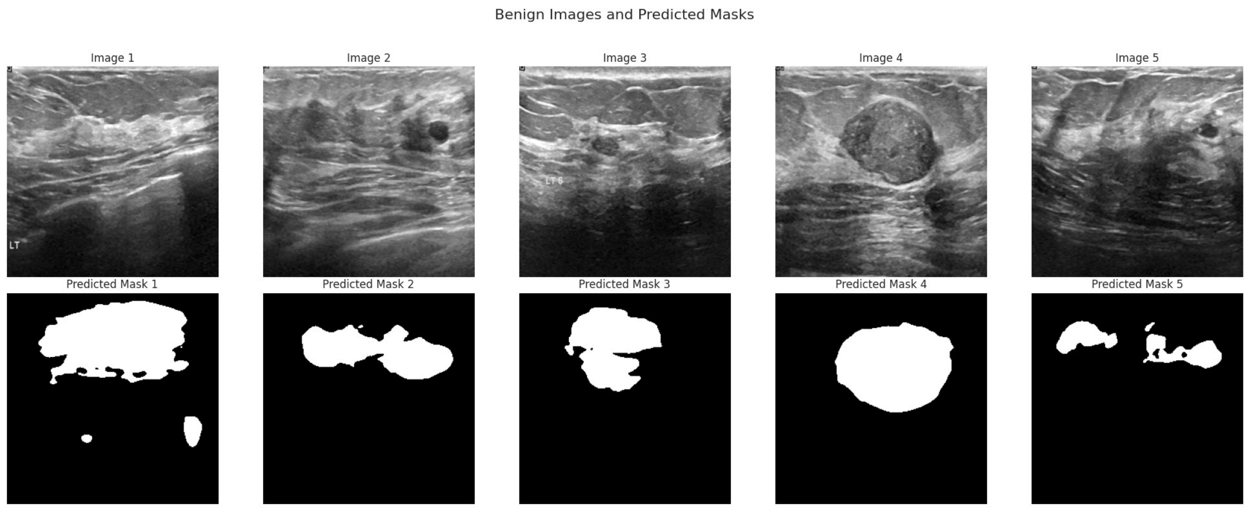
```
        return out

# Build the Hybrid Attention Network
def build_model(input_shape=(256, 256, 3)):
    base_model = DenseNet121(weights='imagenet', include_top=False,
input_shape=input_shape)
    encoder_outputs = [
        base_model.get_layer('conv1_relu').output,  # (128, 128, 64)
        base_model.get_layer('pool2_relu').output,  # (64, 64, 256)
        base_model.get_layer('pool3_relu').output,  # (32, 32, 512)
        base_model.get_layer('pool4_relu').output,  # (16, 16, 1024)
        base_model.get_layer('relu').output        # (8, 8, 1024)
    ]

    x = encoder_outputs[-1]
    x = TSALayer(filters=1024)(x)
    x = GSALayer(filters=1024)(x)

    # Decoder with additional upsampling to reach 256x256
    for i, filters in enumerate([512, 256, 128, 64]):
        x = UpSampling2D(size=(2, 2), interpolation='bilinear')(x)
        skip = encoder_outputs[3 - i]
        skip = SFEB(skip, filters)
        x = Concatenate()([x, skip])
        x = Conv2D(filters, 3, padding='same', activation='relu')(x)
        x = BatchNormalization()(x)
        x = Conv2D(filters, 3, padding='same', activation='relu')(x)
        x = BatchNormalization()(x)

    # Additional upsampling to match 256x256
    x = UpSampling2D(size=(2, 2), interpolation='bilinear')(x)
    x = Conv2D(64, 3, padding='same', activation='relu')(x)
    x = BatchNormalization()(x)

    output = Conv2D(1, 1, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=output)
    return model

def load_data(df_balanced, img_size=(256, 256)):
    images = []
    masks = []

    for _, row in df_balanced.iterrows():
        try:
            img = Image.open(row['image_path']).convert('RGB')
            img = img.resize(img_size)
            img = np.array(img) / 255.0
            mask = Image.open(row['mask_path']).convert('L')
            mask = mask.resize(img_size)
```

```
            mask = np.array(mask) / 255.0
            mask = (mask > 0.5).astype(np.float32)
            mask = np.expand_dims(mask, axis=-1)
            images.append(img)
            masks.append(mask)
        except Exception as e:
            print(f"Error loading image/mask for {row['image_path']}:
{e}")
            continue

    return np.array(images), np.array(masks)

def split_dataset(df_balanced):
    train_df = df_balanced.sample(frac=0.8, random_state=42)
    test_df = df_balanced.drop(train_df.index)
    return train_df, test_df

def visualize_predictions_by_category(df_balanced, model,
num_samples=5):
    tumor_types = ['benign', 'malignant', 'normal']
    for tumor_type in tumor_types:
        category_df = df_balanced[df_balanced['tumor_type'] ==
tumor_type].sample(n=min(num_samples,
len(df_balanced[df_balanced['tumor_type'] == tumor_type])),
random_state=42)
        if len(category_df) == 0:
            print(f"No samples found for {tumor_type}")
            continue
        X, y_true = load_data(category_df)
        y_pred = model.predict(X, batch_size=4)
        y_pred = (y_pred > 0.5).astype(np.float32)

        fig, axes = plt.subplots(2, len(category_df),
figsize=(len(category_df) * 4, 8))
        fig.suptitle(f'{tumor_type.capitalize()} Images and Predicted
Masks', fontsize=16)

        for i in range(len(category_df)):
            axes[0, i].imshow(X[i])
            axes[0, i].set_title(f'Image {i+1}')
            axes[0, i].axis('off')
            axes[1, i].imshow(y_pred[i, :, :, 0], cmap='gray')
            axes[1, i].set_title(f'Predicted Mask {i+1}')
            axes[1, i].axis('off')

        plt.tight_layout(rect=[0, 0, 1, 0.95])
        plt.show()

if __name__ == '__main__':
```

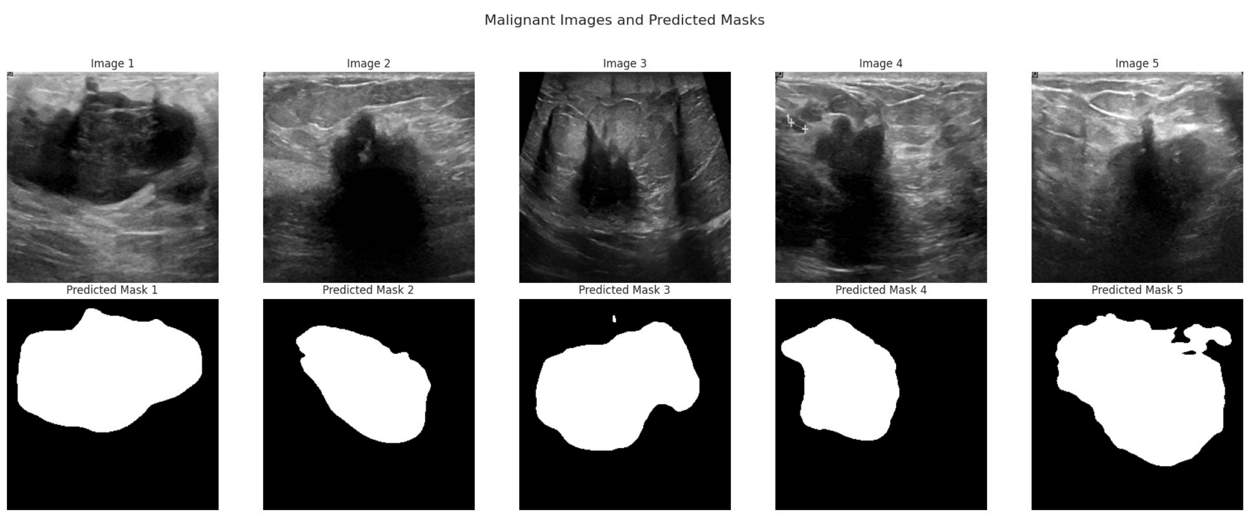
```
    print("Class distribution:")
    print(df_balanced['tumor_type'].value_counts())

    train_df, test_df = split_dataset(df_balanced)

    X_train, y_train = load_data(train_df)
    X_test, y_test = load_data(test_df)

    model = build_model()
    model.compile(optimizer=Adam(learning_rate=0.001),
loss=hybrid_loss,
                 metrics=['accuracy',
tf.keras.metrics.MeanIoU(num_classes=2)])

    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.25,
patience=4, min_lr=1e-6)
    early_stopping = EarlyStopping(monitor='val_loss', patience=2,
restore_best_weights=True)

    try:
        history = model.fit(
            X_train, y_train,
            validation_split=0.2,
            batch_size=4,
            epochs=5,
            callbacks=[reduce_lr, early_stopping],
            verbose=1
        )

        evaluation = model.evaluate(X_test, y_test, batch_size=4,
verbose=1)
        print(f"Test Loss: {evaluation[0]:.4f}")
        print(f"Test Accuracy: {evaluation[1]:.4f}")
        print(f"Test IoU: {evaluation[2]:.4f}")

        visualize_predictions_by_category(df_balanced, model,
num_samples=5)

    except Exception as e:
        print(f"Error during training or evaluation: {e}")

Class distribution:
tumor_type
normal       437
benign       437
malignant    437
Name: count, dtype: int64
Epoch 1/5
```

```
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1751689878.901433     133 service.cc:148] XLA service
0x7f997c004550 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
I0000 00:00:1751689878.903156     133 service.cc:156]   StreamExecutor
device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1751689878.903181     133 service.cc:156]   StreamExecutor
device (1): Tesla T4, Compute Capability 7.5
I0000 00:00:1751689889.971311     133 cuda_dnn.cc:529] Loaded cuDNN
version 90300
E0000 00:00:1751689908.474346     133 gpu_timer.cc:82] Delay kernel
timed out: measured time has sub-optimal accuracy. There may be a
missing warmup execution, please investigate in Nsight Systems.
E0000 00:00:1751689908.748960     133 gpu_timer.cc:82] Delay kernel
timed out: measured time has sub-optimal accuracy. There may be a
missing warmup execution, please investigate in Nsight Systems.
I0000 00:00:1751690004.690927     133 device_compiler.h:188] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

209/210 ━━━━━━━━━━━━━━━━━━━━ 0s 218ms/step - accuracy: 0.7432 - loss:
7.7835 - mean_io_u_1: 0.4634

E0000 00:00:1751690074.102887     133 gpu_timer.cc:82] Delay kernel
timed out: measured time has sub-optimal accuracy. There may be a
missing warmup execution, please investigate in Nsight Systems.
E0000 00:00:1751690074.374450     133 gpu_timer.cc:82] Delay kernel
timed out: measured time has sub-optimal accuracy. There may be a
missing warmup execution, please investigate in Nsight Systems.

210/210 ━━━━━━━━━━━━━━━━━━━━ 459s 1s/step - accuracy: 0.7441 - loss:
7.7780 - mean_io_u_1: 0.4634 - val_accuracy: 0.5100 - val_loss:
49.3701 - val_mean_io_u_1: 0.3440 - learning_rate: 0.0010
Epoch 2/5
210/210 ━━━━━━━━━━━━━━━━━━━━ 49s 231ms/step - accuracy: 0.9203 - loss:
6.6040 - mean_io_u_1: 0.4632 - val_accuracy: 0.8340 - val_loss: 6.6127
- val_mean_io_u_1: 0.4899 - learning_rate: 0.0010
Epoch 3/5
210/210 ━━━━━━━━━━━━━━━━━━━━ 48s 228ms/step - accuracy: 0.9340 - loss:
6.6647 - mean_io_u_1: 0.4659 - val_accuracy: 0.8733 - val_loss: 6.6093
- val_mean_io_u_1: 0.5358 - learning_rate: 0.0010
Epoch 4/5
210/210 ━━━━━━━━━━━━━━━━━━━━ 48s 227ms/step - accuracy: 0.9414 - loss:
6.2724 - mean_io_u_1: 0.4642 - val_accuracy: 0.9333 - val_loss: 8.0397
- val_mean_io_u_1: 0.4646 - learning_rate: 0.0010
Epoch 5/5
210/210 ━━━━━━━━━━━━━━━━━━━━ 48s 228ms/step - accuracy: 0.9342 - loss:
6.3593 - mean_io_u_1: 0.4644 - val_accuracy: 0.8512 - val_loss: 6.2562
- val_mean_io_u_1: 0.4993 - learning_rate: 0.0010
```

```
66/66 ━━━━━━━━━━━━━━━━━━━━ 4s 61ms/step - accuracy: 0.8564 - loss:
6.0682 - mean_io_u_1: 0.4951
Test Loss: 6.6279
Test Accuracy: 0.8547
Test IoU: 0.4974
2/2 ━━━━━━━━━━━━━━━━━━━━ 34s 20s/step
```




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
```




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
```

Normal Images and Predicted Masks