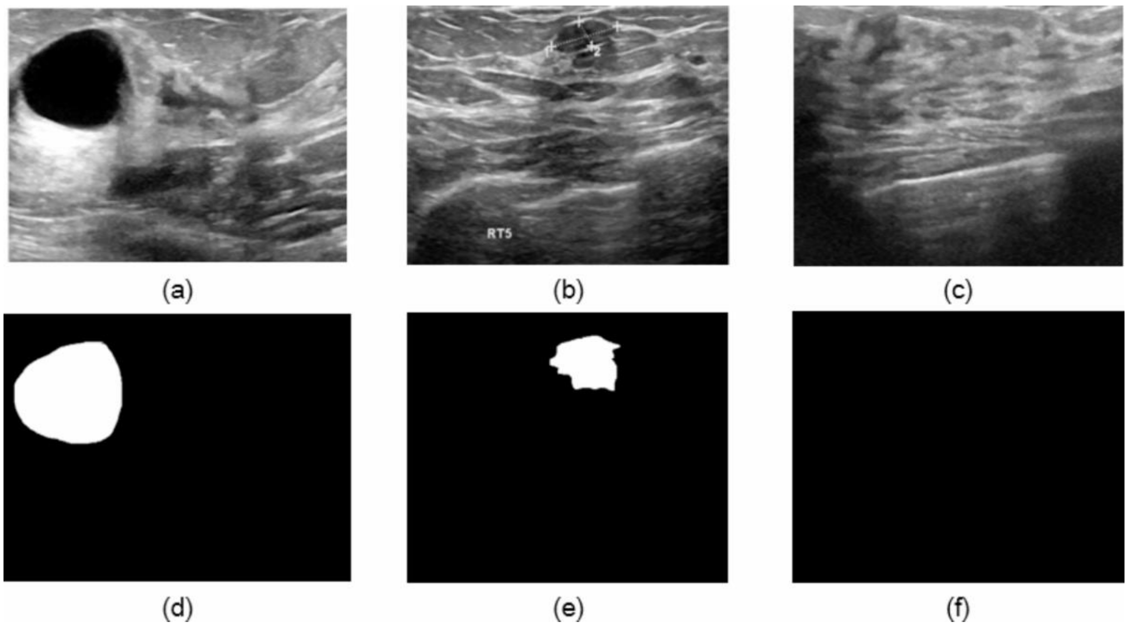
Breast Cancer Segmentation using HCMA - Unet Novel Architecture



```
import numpy as np
import pandas as pd

base_path = "/kaggle/input/breast-ultrasound-images-
dataset/Dataset_BUSI_with_GT/"
categories = ["benign","malignant", "normal"]

data = []

for category in categories:
    category_path = os.path.join(base_path, category)
    files = sorted(os.listdir(category_path))

    for file in files:
        if "_mask" in file:
            continue

        img_path = os.path.join(category_path, file)
        mask_path = os.path.join(category_path, file.replace(".png",
"_mask.png"))

        data.append([img_path, mask_path, category])

df = pd.DataFrame(data, columns=["Image_Path", "Mask_Path", "Label"])

df.head()
```

```
                                          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...

                                           Mask_Path   Label
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
```

```
df.tail()
```

```
                                            Image_Path  \
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   Label
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
```

```
df.shape
```

```
(780, 3)
```

```
df.columns
```

```
Index(['Image_Path', 'Mask_Path', 'Label'], dtype='object')
```

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

```
0
```

```
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   Label       780 non-null    object
```

```
dtypes: object(3)
memory usage: 18.4+ KB
```

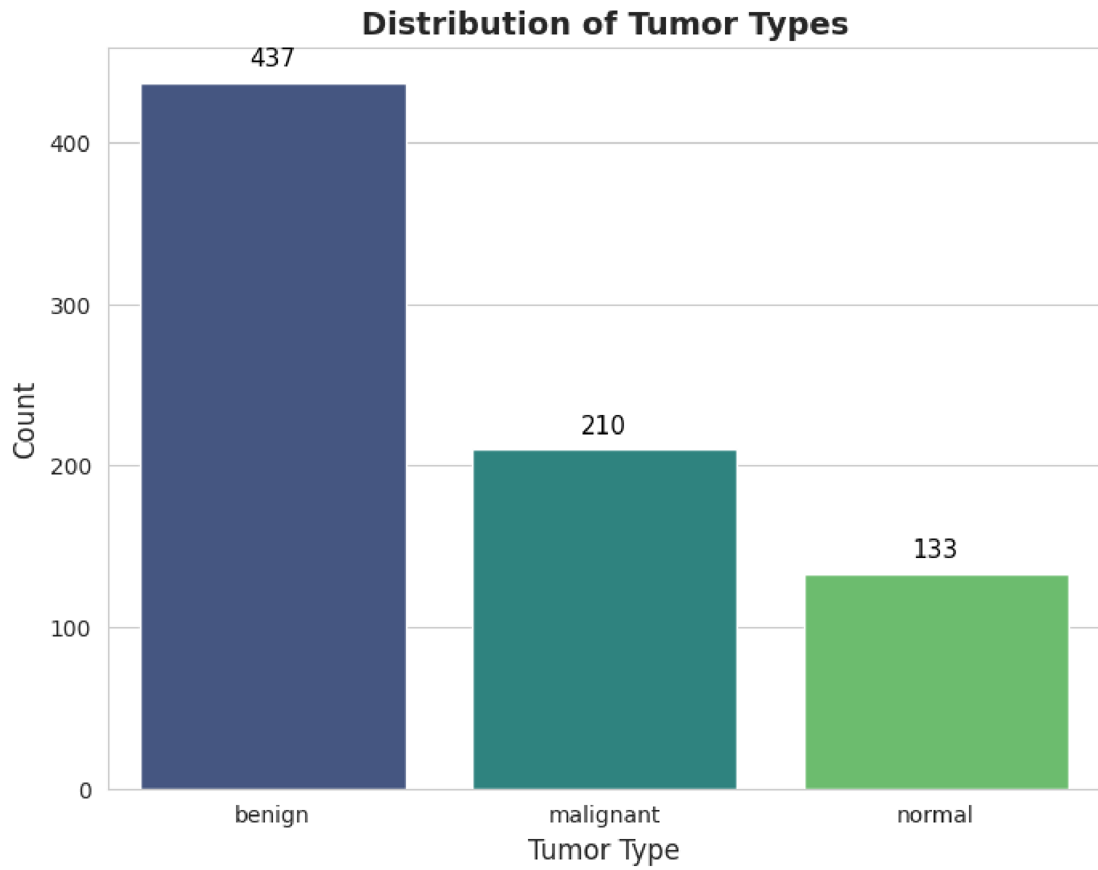
```python
df['Label'].unique()
```

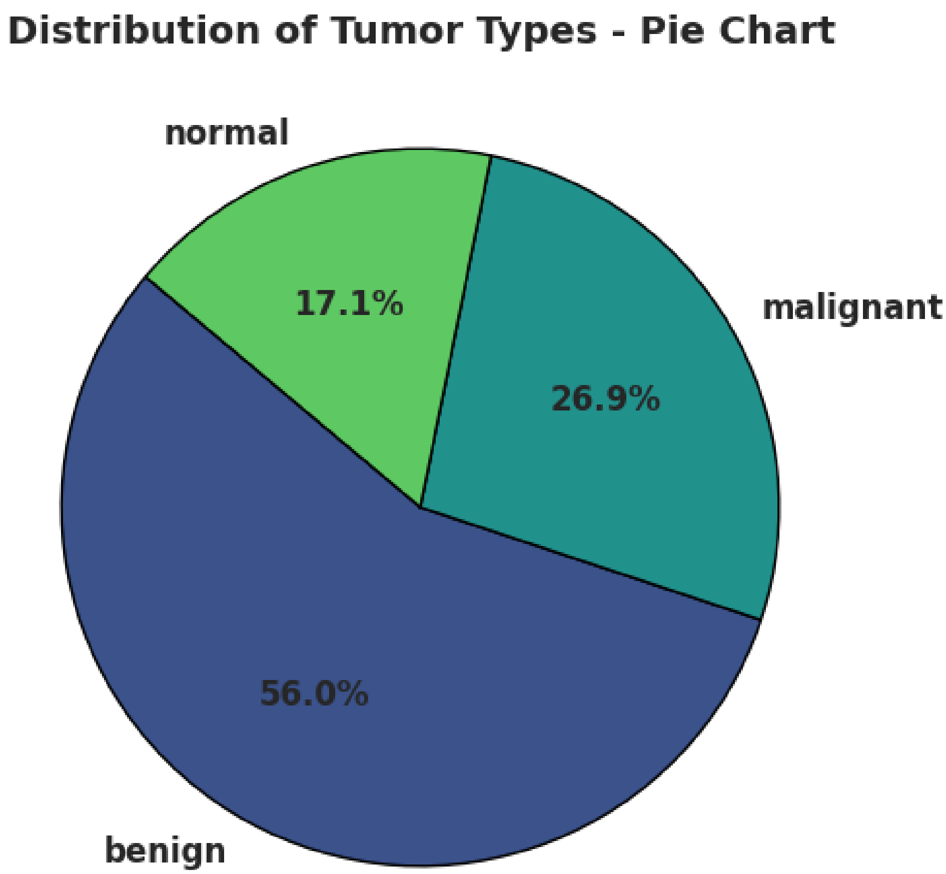
```
array(['benign', 'malignant', 'normal'], dtype=object)
```

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

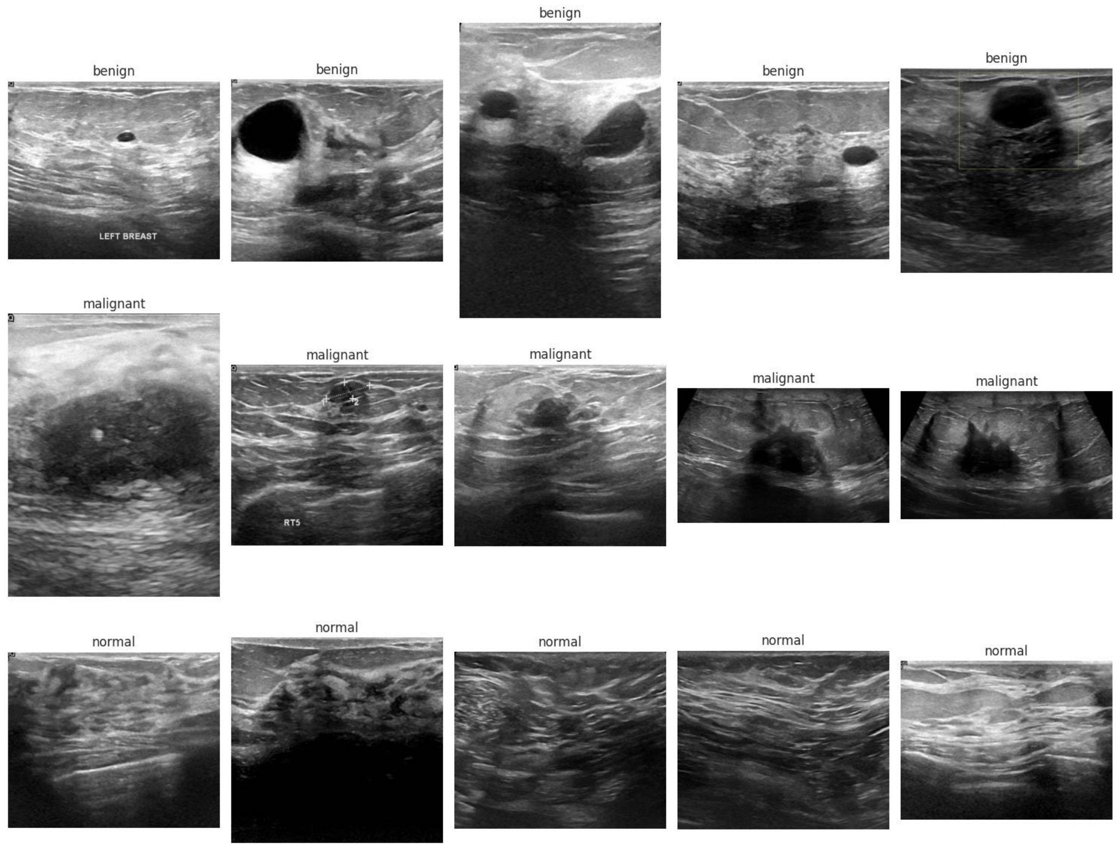
```
Label
benign       437
malignant    210
normal       133
Name: count, dtype: int64
```

```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="Label", palette="viridis", ax=ax)

ax.set_title("Distribution of Tumor 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["Label"].value_counts()

fig, ax = plt.subplots(figsize=(8, 6))
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 Tumor Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```

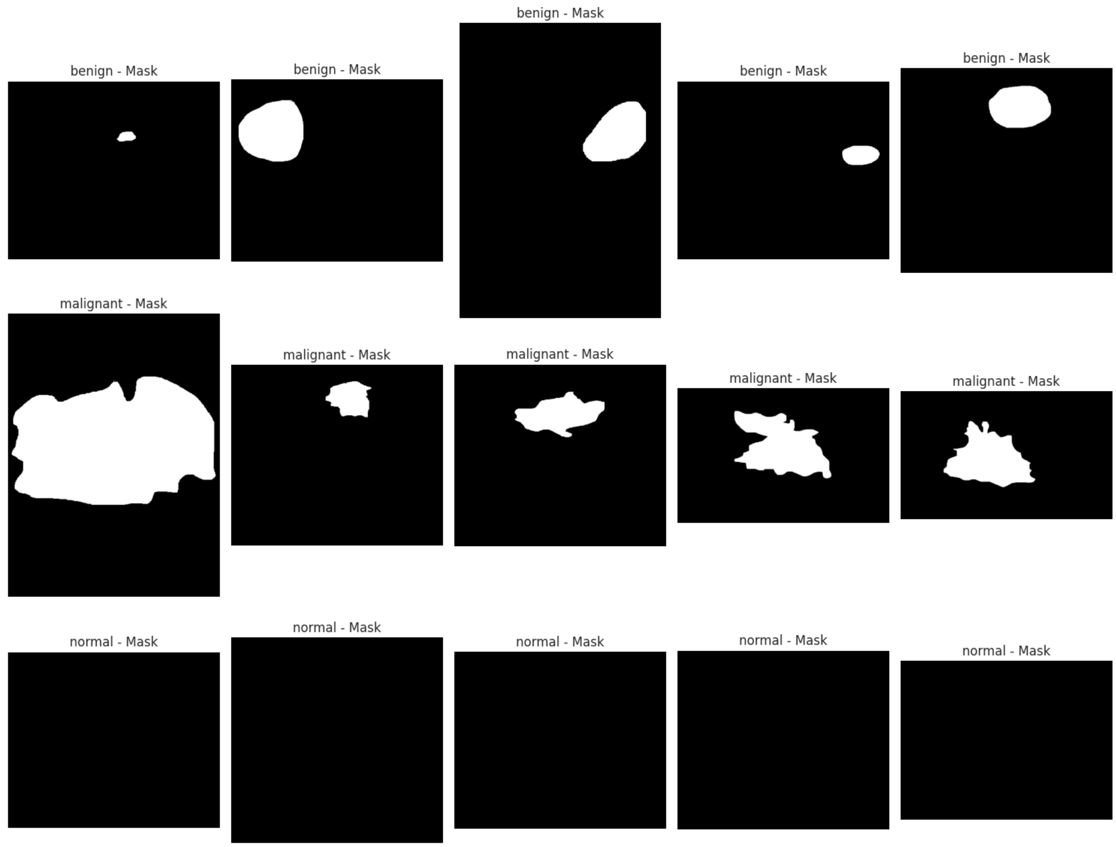

Distribution of Tumor Types
437
210
133
400
300
200
100
0
Count
benign
malignant
normal
Tumor Type




```python
import cv2

num_images = 5
plt.figure(figsize=(15, 12))

for i, category in enumerate(categories):
    category_images = df[df["Label"] ==
category]["Image_Path"].iloc[:num_images]

    for j, img_path in enumerate(category_images):
        img = cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB)
        plt.subplot(len(categories), num_images, i * num_images + j + 1)
        plt.imshow(img)
        plt.axis("off")
        plt.title(category)

plt.tight_layout()
plt.show()
```

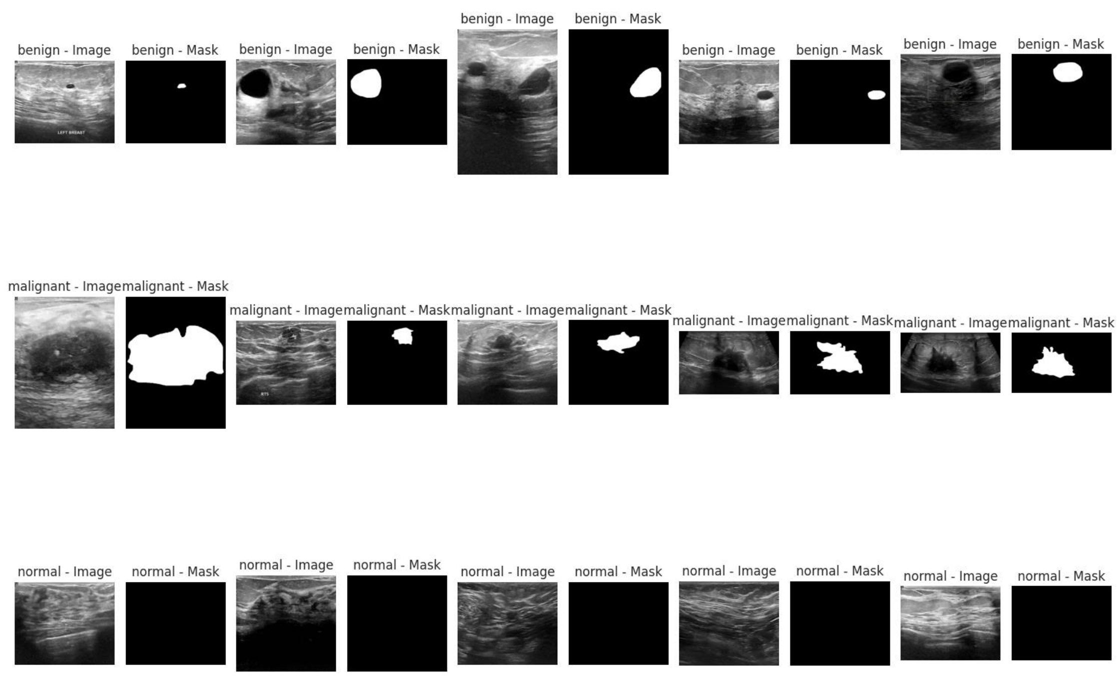



```
num_images = 5
plt.figure(figsize=(15, 12))

for i, category in enumerate(categories):
    category_masks = df[df["Label"] ==
category]["Mask_Path"].iloc[:num_images]

    for j, mask_path in enumerate(category_masks):
        mask = cv2.cvtColor(cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE),
cv2.COLOR_GRAY2RGB)

        plt.subplot(len(categories), num_images, i * num_images + j + 1)
        plt.imshow(mask)
        plt.axis("off")
        plt.title(f"{category} - Mask")

plt.tight_layout()
plt.show()
```




```
num_images = 5
plt.figure(figsize=(15, 12))

for i, category in enumerate(categories):
    category_images = df[df["Label"] ==
category]["Image_Path"].iloc[:num_images]
    category_masks = df[df["Label"] ==
category]["Mask_Path"].iloc[:num_images]

    for j, (img_path, mask_path) in enumerate(zip(category_images,
category_masks)):
        img = cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB)
        mask = cv2.cvtColor(cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE),
cv2.COLOR_GRAY2RGB)

        plt.subplot(len(categories), num_images * 2, i * num_images * 2 + j *
2 + 1)
        plt.imshow(img)
        plt.axis("off")
        plt.title(f"{category} - Image")

        plt.subplot(len(categories), num_images * 2, i * num_images * 2 + j *
2 + 2)
        plt.imshow(mask)
```

```
        plt.axis("off")
        plt.title(f"{category} - Mask")

plt.tight_layout()
plt.show()
```




```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

df['category_encoded'] = label_encoder.fit_transform(df['Label'])
```

```
from imblearn.over_sampling import RandomOverSampler

X = df[['Image_Path', 'Mask_Path']]
y = df['category_encoded']

ros = RandomOverSampler(random_state=42)

X_resampled, y_resampled = ros.fit_resample(X, y)

df_resampled = pd.DataFrame(X_resampled, columns=['Image_Path', 'Mask_Path'])
df_resampled['category_encoded'] = y_resampled
```

```
df_resampled.columns
```

```
Index(['Image_Path', 'Mask_Path', 'category_encoded'], dtype='object')
```

```
df_resampled['category_encoded'].value_counts()
```

```
category_encoded
0    437
1    437
2    437
Name: count, dtype: int64
```

```
df_resampled['category_encoded'] =
df_resampled['category_encoded'].astype(str)

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers

import warnings
warnings.filterwarnings("ignore")

print ('check')
```

```
check
```

```
train_df_new, temp_df_new = train_test_split(
    df_resampled,
    train_size=0.8,
    shuffle=True,
    random_state=42,
    stratify=df_resampled['category_encoded']
)

valid_df_new, test_df_new = train_test_split(
    temp_df_new,
    test_size=0.5,
    shuffle=True,
    random_state=42,
    stratify=temp_df_new['category_encoded']
)

import tensorflow as tf
import numpy as np
import os
import cv2
from tensorflow.keras.layers import Conv3D, Conv3DTranspose,
BatchNormalization, Activation, Add, Input
from tensorflow.keras.models import Model
```

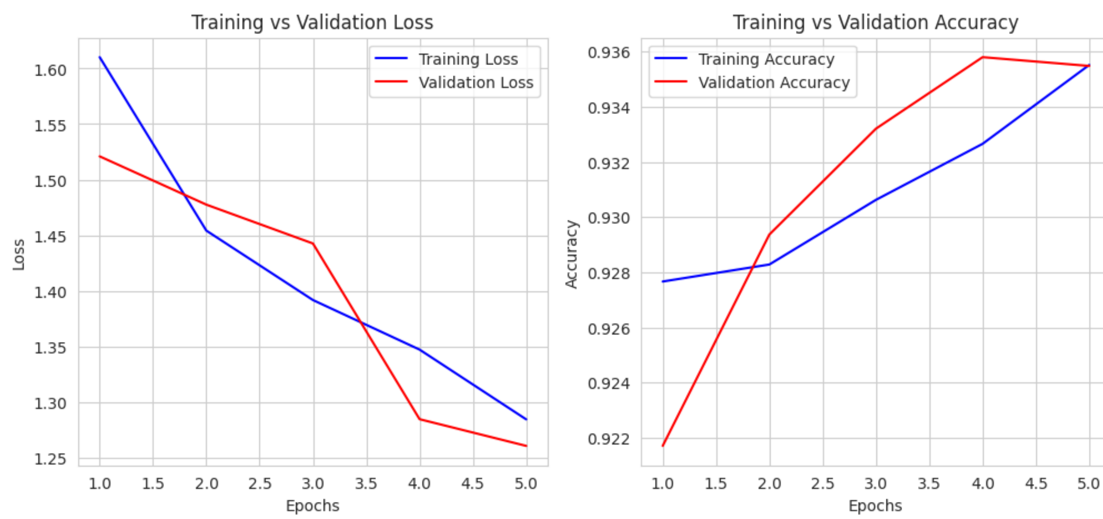
```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
from sklearn.model_selection import train_test_split

def load_images_and_masks(image_paths, mask_paths):
    images, masks = [], []
    for img_path, mask_path in zip(image_paths, mask_paths):
        image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        image = cv2.resize(image, (128, 128)) / 255.0
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        mask = cv2.resize(mask, (128, 128)) / 255.0
        images.append(image)
        masks.append(mask)
    return np.array(images)[..., np.newaxis], np.array(masks)[...,
np.newaxis]

X_train, Y_train = load_images_and_masks(train_df_new['Image_Path'].values,
train_df_new['Mask_Path'].values)
X_val, Y_val = load_images_and_masks(valid_df_new['Image_Path'].values,
valid_df_new['Mask_Path'].values)

X_test, Y_test = load_images_and_masks(test_df_new['Image_Path'].values,
test_df_new['Mask_Path'].values)

print(f"Training data shape: {X_train.shape}, Training mask shape:
{Y_train.shape}")
print(f"Validation data shape: {X_val.shape}, Validation mask shape:
{Y_val.shape}")
print(f"Testing data shape: {X_test.shape}, Testing mask shape:
{Y_test.shape}")

Training data shape: (1048, 128, 128, 1), Training mask shape: (1048, 128,
128, 1)
Validation data shape: (131, 128, 128, 1), Validation mask shape: (131, 128,
128, 1)
Testing data shape: (132, 128, 128, 1), Testing mask shape: (132, 128, 128,
1)

import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Conv2DTranspose,
LayerNormalization, DepthwiseConv2D, concatenate, ReLU, Softmax
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import AdamW
import numpy as np

class ResBlock(tf.keras.layers.Layer):
    def __init__(self, filters, kernel_size=3, strides=1, padding='same'):
        super(ResBlock, self).__init__()
        self.conv1 = Conv2D(filters, kernel_size, strides=strides,
padding=padding)
        self.conv2 = Conv2D(filters, kernel_size, strides=strides,
```

```
padding=padding)
        self.relu = ReLU()
        self.match_dimensions = Conv2D(filters, kernel_size=1,
padding='same')

    def call(self, inputs):
        residual = inputs
        x = self.conv1(inputs)
        x = self.relu(x)
        x = self.conv2(x)
        residual = self.match_dimensions(residual)
        x += residual
        return self.relu(x)

class VSSB(tf.keras.layers.Layer):
    def __init__(self, filters):
        super(VSSB, self).__init__()
        self.dwconv = DepthwiseConv2D(kernel_size=11, strides=1,
padding='same')
        self.ln = LayerNormalization()
        self.linear = Conv2D(filters, kernel_size=1, strides=1,
padding='same')

    def call(self, inputs):
        x = self.dwconv(inputs)
        x = self.ln(x)
        x = self.linear(x)
        return x

class ISSA(tf.keras.layers.Layer):
    def __init__(self, filters):
        super(ISSA, self).__init__()
        self.ln = LayerNormalization()
        self.linear = Conv2D(filters, kernel_size=1, strides=1,
padding='same')

    def call(self, inputs):
        x = self.ln(inputs)
        x = self.linear(x)
        return x

class MISM(tf.keras.layers.Layer):
    def __init__(self, filters):
        super(MISM, self).__init__()
        self.vssb = VSSB(filters)
        self.issa = ISSA(filters)

    def call(self, inputs):
        x = self.vssb(inputs)
```

```
        x = self.issa(x)
        return x

class DenseBlock(tf.keras.layers.Layer):
    def __init__(self, filters):
        super(DenseBlock, self).__init__()
        self.conv1 = Conv2D(filters, kernel_size=3, strides=1,
padding='same')
        self.conv2 = Conv2D(filters, kernel_size=1, strides=1,
padding='same')
        self.conv3 = Conv2D(filters, kernel_size=1, strides=1,
padding='same')

    def call(self, inputs):
        x1 = self.conv1(inputs)
        x2 = self.conv2(concatenate([inputs, x1]))
        x3 = self.conv3(concatenate([inputs, x1, x2]))
        return x3

class HCMA_UNet(Model):
    def __init__(self):
        super(HCMA_UNet, self).__init__()
        # Encoder
        self.encoder1 = ResBlock(64)
        self.encoder2 = ResBlock(128)
        self.encoder3 = ResBlock(256)
        # Middle
        self.mism = MISM(256)
        self.dense = DenseBlock(256)
        # Decoder
        self.decoder1 = Conv2DTranspose(128, kernel_size=2, strides=2,
padding='same')
        self.decoder2 = Conv2DTranspose(64, kernel_size=2, strides=2,
padding='same')
        self.final_conv = Conv2D(1, kernel_size=1, activation='sigmoid')

    def call(self, inputs):

        x1 = self.encoder1(inputs)
        x2 = self.encoder2(x1)
        x3 = self.encoder3(x2)

        x = self.mism(x3)
        x = self.dense(x)

        x = self.decoder1(x)
        x = self.decoder2(x)
        x = self.final_conv(x)
```

```
        x = tf.image.resize(x, (128, 128))
        return x

class FRLoss(tf.keras.losses.Loss):
    def __init__(self, lambda_fr=5):
        super(FRLoss, self).__init__()
        self.lambda_fr = lambda_fr

    def call(self, y_true, y_pred):

        intersection = tf.reduce_sum(y_true * y_pred)
        union = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred)
        dice_loss = 1 - (2 * intersection + 1e-7) / (union + 1e-7)

        ce_loss = tf.keras.losses.binary_crossentropy(y_true, y_pred)

        fr_loss = self.lambda_fr * tf.reduce_mean(tf.abs(y_true - y_pred))

        return dice_loss + ce_loss + fr_loss

model = HCMA_UNet()
model.compile(optimizer=AdamW(learning_rate=1e-4), loss=FRLoss(),
metrics=['accuracy'])

model.compile(optimizer=AdamW(learning_rate=1e-4), loss=FRLoss(),
metrics=['accuracy'])

history = model.fit(X_train, Y_train, validation_data=(X_val, Y_val),
batch_size=2, epochs=5)

Epoch 1/5
524/524 ━━━━━━━━━━━━━━━━━━━━ 121s 180ms/step - accuracy: 0.9245 - loss:
1.6959 - val_accuracy: 0.9217 - val_loss: 1.5210
Epoch 2/5
524/524 ━━━━━━━━━━━━━━━━━━━━ 85s 162ms/step - accuracy: 0.9252 - loss: 1.4871
- val_accuracy: 0.9294 - val_loss: 1.4776
Epoch 3/5
524/524 ━━━━━━━━━━━━━━━━━━━━ 85s 163ms/step - accuracy: 0.9298 - loss: 1.4165
- val_accuracy: 0.9332 - val_loss: 1.4426
Epoch 4/5
524/524 ━━━━━━━━━━━━━━━━━━━━ 85s 163ms/step - accuracy: 0.9367 - loss: 1.3295
- val_accuracy: 0.9358 - val_loss: 1.2845
Epoch 5/5
524/524 ━━━━━━━━━━━━━━━━━━━━ 86s 163ms/step - accuracy: 0.9355 - loss: 1.2903
- val_accuracy: 0.9355 - val_loss: 1.2605

loss = history.history['loss']
val_loss = history.history['val_loss']
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
```

```
epochs = range(1, len(loss) + 1)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, loss, 'b-', label='Training Loss')
plt.plot(epochs, val_loss, 'r-', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training vs Validation Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, accuracy, 'b-', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'r-', label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy')
plt.legend()

plt.show()
```

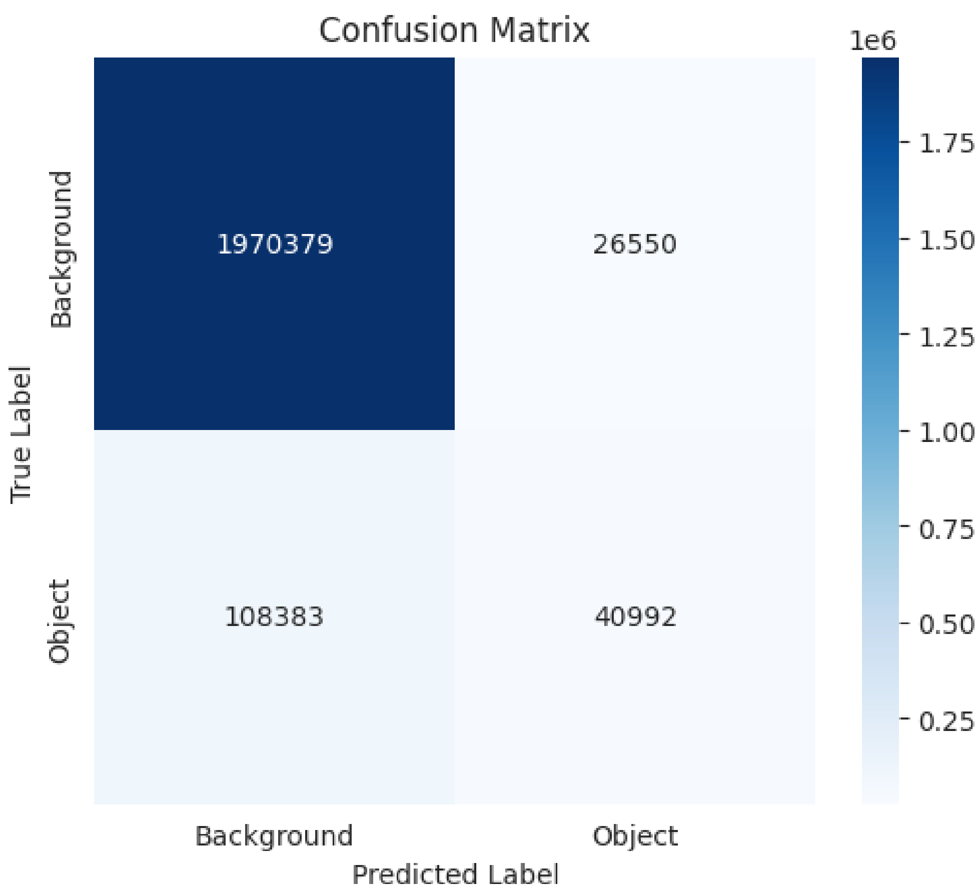



```
y_pred = model.predict(X_val)

y_pred_bin = (y_pred > 0.5).astype(np.uint8)

y_true_flat = Y_val.flatten()
y_pred_flat = y_pred_bin.flatten()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 62s 3s/step
```

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
```

```
y_pred_bin = (y_pred > 0.5).astype(np.uint8)

y_true_flat = Y_val.flatten().astype(np.uint8)
y_pred_flat = y_pred_bin.flatten()

cm = confusion_matrix(y_true_flat, y_pred_flat)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Background",
"Object"], yticklabels=["Background", "Object"])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

print("Classification Report:\n", classification_report(y_true_flat,
y_pred_flat, target_names=["Background", "Object"]))
```




```
Classification Report:
              precision    recall  f1-score   support

  Background       0.95      0.99      0.97   1996929
```

Object	0.61	0.27	0.38	149375
accuracy			0.94	2146304
macro avg	0.78	0.63	0.67	2146304
weighted avg	0.92	0.94	0.93	2146304