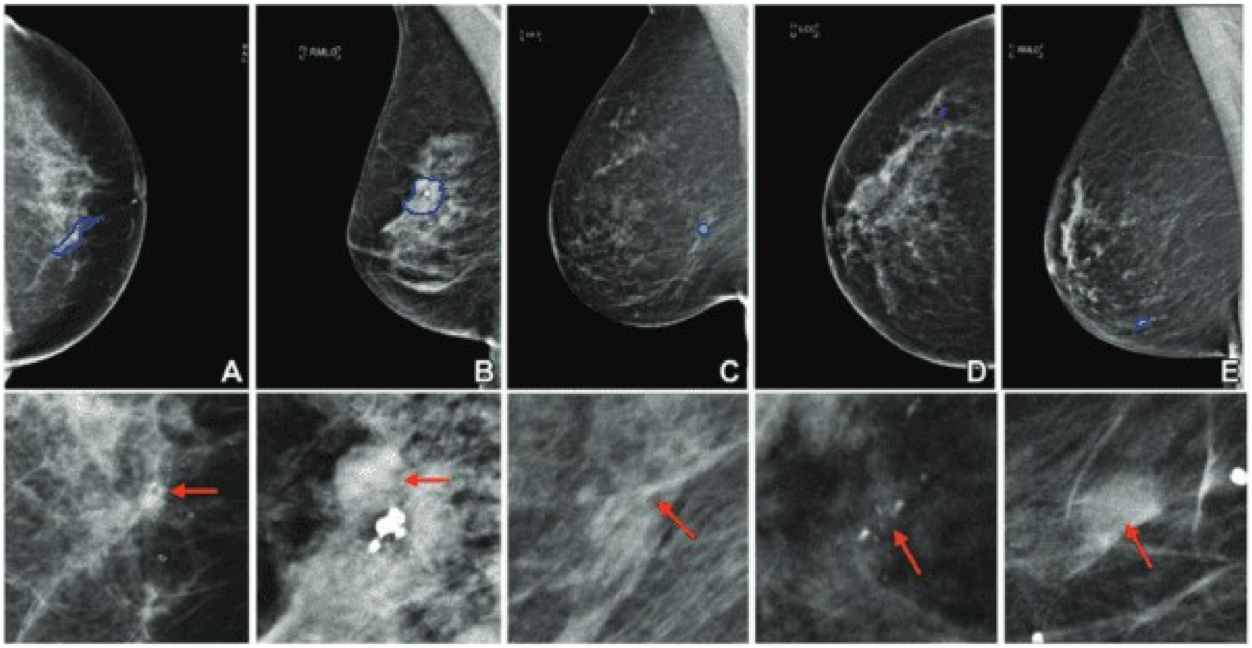
Multimodal Breast Cancer Analysis




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

base_path = "/kaggle/input/breast-cancer-msi-multimodal-image-dataset/MultiModel Breast Cancer MSI Dataset/Chest_XRay_MSI/"
categories = ["Malignant", "Normal"]

base_path_1 = "/kaggle/input/breast-cancer-msi-multimodal-image-dataset/MultiModel Breast Cancer MSI Dataset/Histopathological_MSI/"
categories = ["benign", "malignant"]

base_path_2 = "/kaggle/input/breast-cancer-msi-multimodal-image-dataset/MultiModel Breast Cancer MSI Dataset/Ultrasound Images_MSI/"
categories = ["benign", "malignant"]

image_paths = []
labels = []

for category in categories:
    category_path = os.path.join(base_path, category)
    for image_name in os.listdir(category_path):
        image_path = os.path.join(category_path, image_name)
        image_paths.append(image_path)
        labels.append(category)
```

```python
df = pd.DataFrame({
    "image_path": image_paths,
    "label": labels
})

image_paths = []
labels = []

for category in categories:
    category_path = os.path.join(base_path_1, category)
    for image_name in os.listdir(category_path):
        image_path = os.path.join(category_path, image_name)
        image_paths.append(image_path)
        labels.append(category)

df1 = pd.DataFrame({
    "image_path": image_paths,
    "label": labels
})

image_paths = []
labels = []

for category in categories:
    category_path = os.path.join(base_path_2, category)
    for image_name in os.listdir(category_path):
        image_path = os.path.join(category_path, image_name)
        image_paths.append(image_path)
        labels.append(category)

df2 = pd.DataFrame({
    "image_path": image_paths,
    "label": labels
})

df
```

```
                                            image_path      label
0    /kaggle/input/breast-cancer-msi-multimodal-ima...  Malignant
1    /kaggle/input/breast-cancer-msi-multimodal-ima...  Malignant
2    /kaggle/input/breast-cancer-msi-multimodal-ima...  Malignant
3    /kaggle/input/breast-cancer-msi-multimodal-ima...  Malignant
4    /kaggle/input/breast-cancer-msi-multimodal-ima...  Malignant
..                                                 ...        ...
995  /kaggle/input/breast-cancer-msi-multimodal-ima...     Normal
996  /kaggle/input/breast-cancer-msi-multimodal-ima...     Normal
997  /kaggle/input/breast-cancer-msi-multimodal-ima...     Normal
998  /kaggle/input/breast-cancer-msi-multimodal-ima...     Normal
999  /kaggle/input/breast-cancer-msi-multimodal-ima...     Normal

[1000 rows x 2 columns]
```

```
df1
```

```
                                             image_path      label
0     /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
1     /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
2     /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
3     /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
4     /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
...                                                 ...        ...
1241  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant
1242  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant
1243  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant
1244  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant
1245  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant

[1246 rows x 2 columns]
```

```
df2
```

```
                                             image_path      label
0    /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
1    /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
2    /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
3    /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
4    /kaggle/input/breast-cancer-msi-multimodal-ima...     benign
..                                                 ...        ...
801  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant
802  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant
803  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant
804  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant
805  /kaggle/input/breast-cancer-msi-multimodal-ima...  malignant

[806 rows x 2 columns]
```

```
df.shape
```

```
(1000, 2)
```

```
df.columns
```

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

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

```
0
```

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

```
image_path    0
label         0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  1000 non-null   object
 1   label       1000 non-null   object
dtypes: object(2)
memory usage: 15.8+ KB
```

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

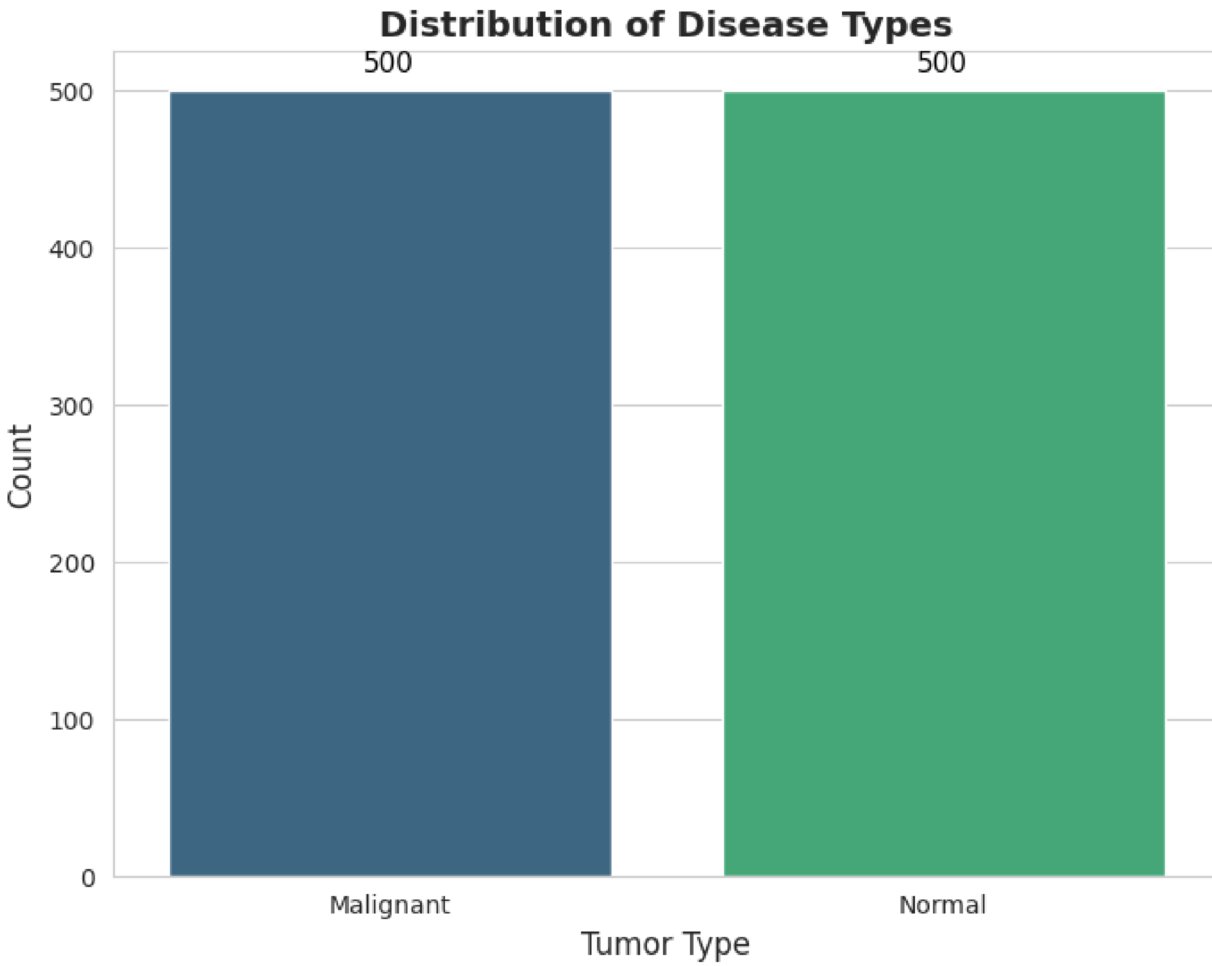
```
array(['Malignant', 'Normal'], dtype=object)
```

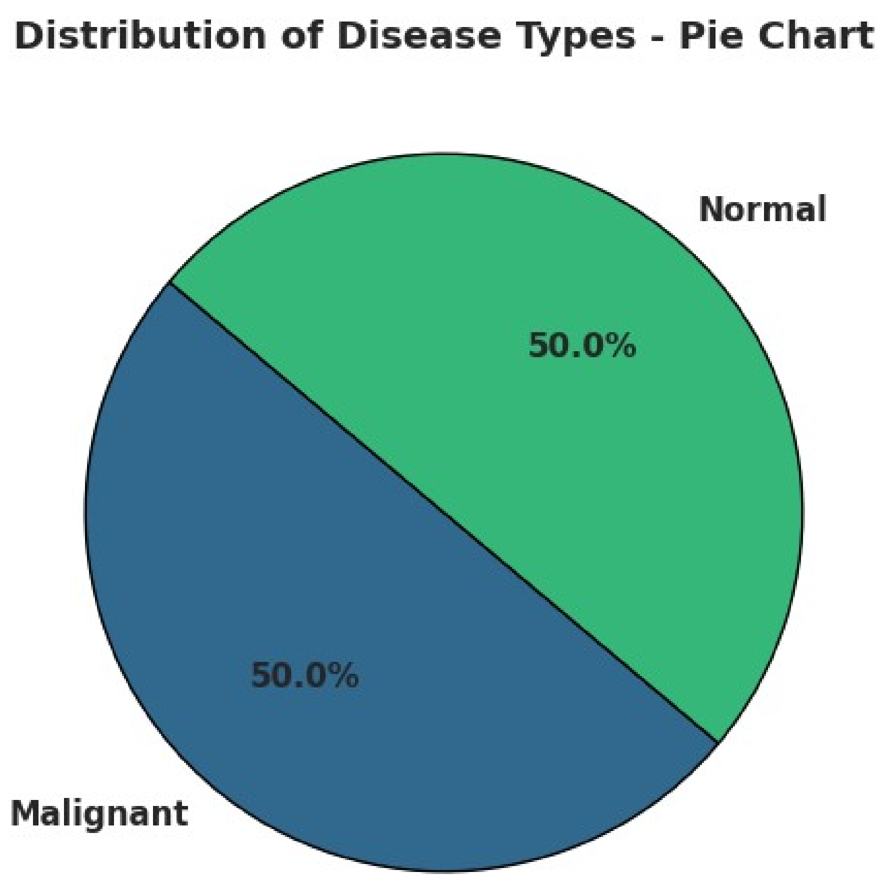
```
df['label'].value_counts()
```

```
label
Malignant    500
Normal       500
Name: count, dtype: int64
```

```
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 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["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,
```

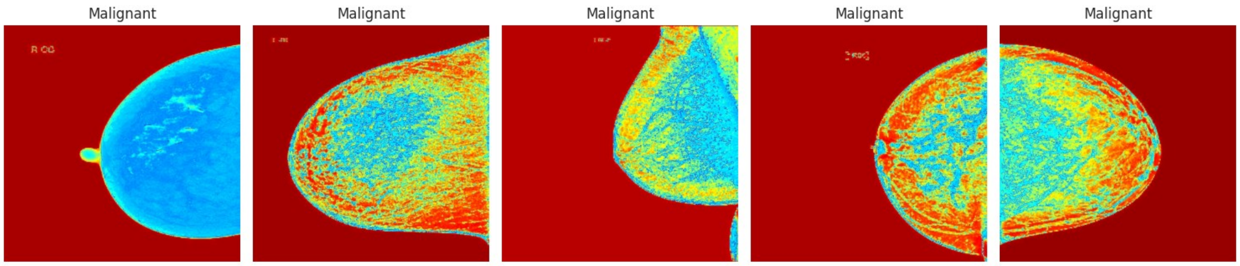
```
'weight': 'bold'},
       wedgeprops={'edgecolor': 'black', 'linewidth': 1})

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

plt.show()
```

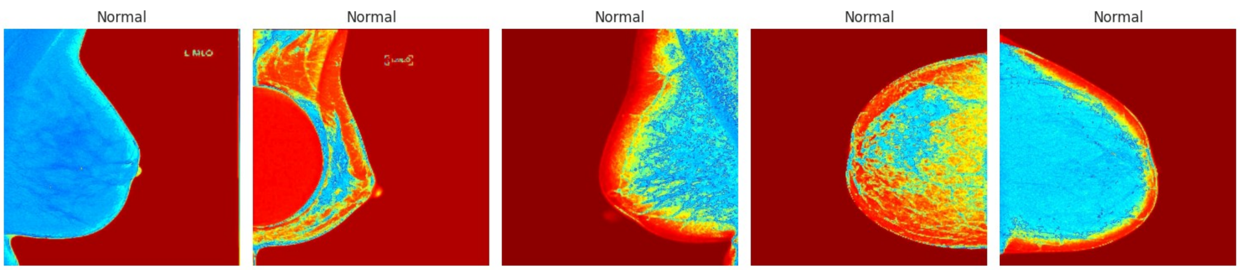





```
import cv2

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.imread(img_path)
        img = cv2.cvtColor(img, 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()
```






```
df1.shape
```

```
(1246, 2)
```

```
df1.columns
```

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

```
df1.duplicated().sum()
```

```
0
```

```
df1.isnull().sum()
```

```
image_path    0
label         0
dtype: int64
```

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1246 entries, 0 to 1245
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  1246 non-null   object
 1   label       1246 non-null   object
```

```
dtypes: object(2)
memory usage: 19.6+ KB
```

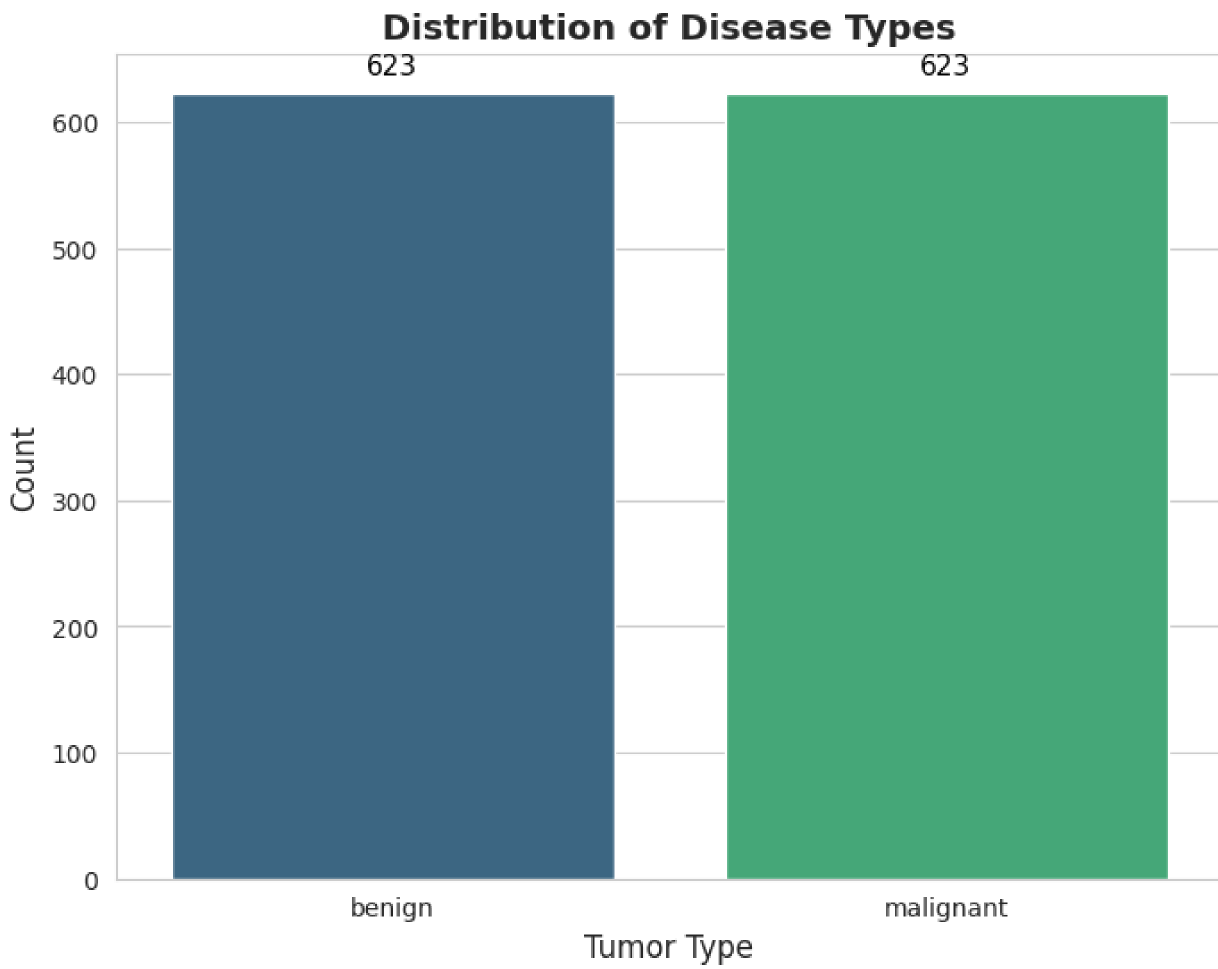
```
df1['label'].unique()
```

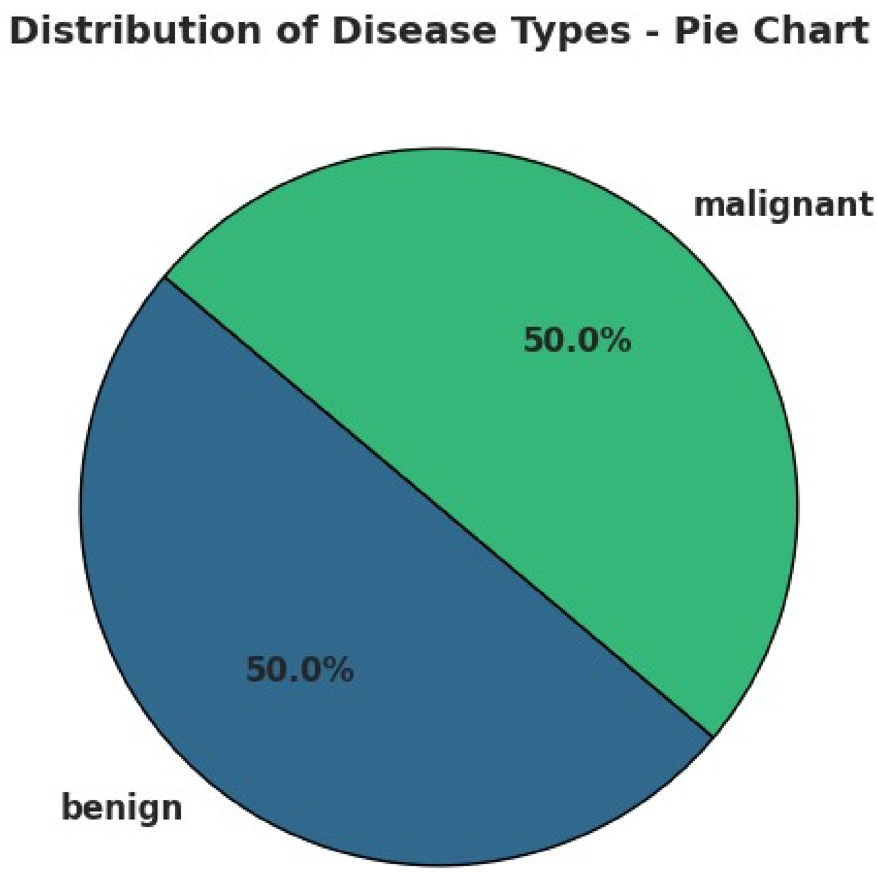
```
array(['benign', 'malignant'], dtype=object)
```

```
df1['label'].value_counts()
```

```
label
benign       623
malignant    623
Name: count, dtype: int64
```

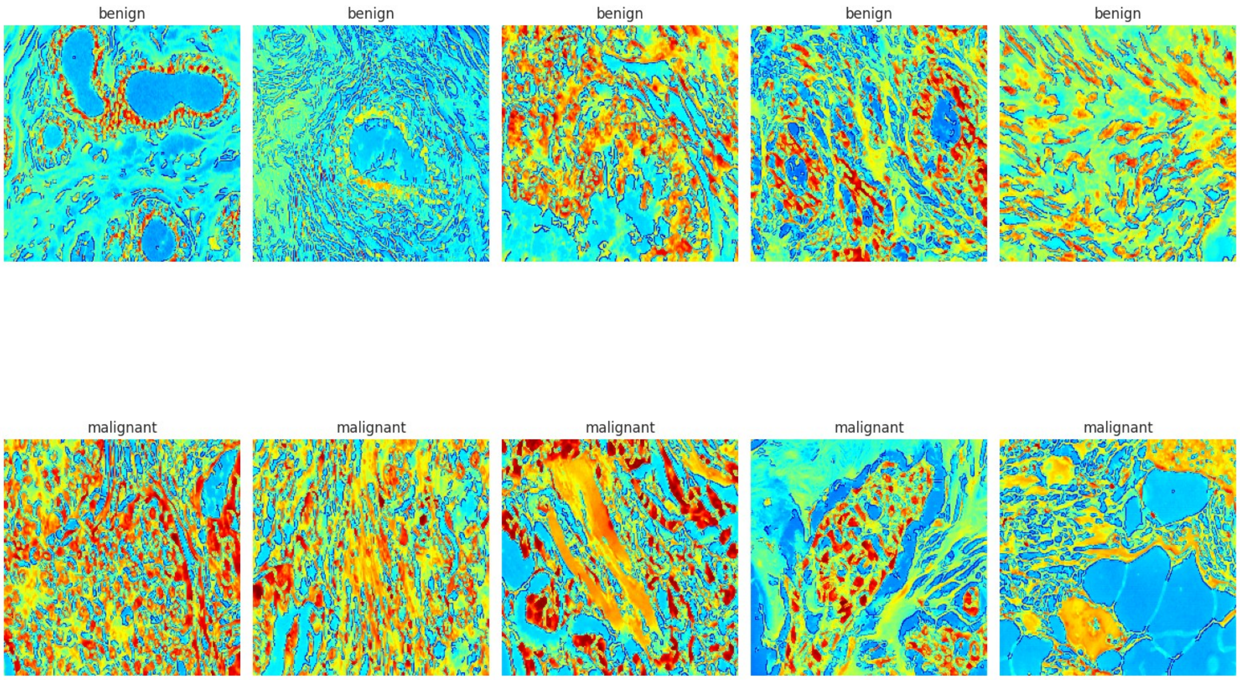
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(8, 6))
sns.countplot(data=df1, x="label", 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 = df1["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 Disease Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution of Disease Types
623
623
600
500
400
300
200
100
0
Count
benign
malignant
Tumor Type




```
import cv2

num_images = 5

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

for i, category in enumerate(categories):
    category_images = df1[df1['label'] == category]
['image_path'].iloc[:num_images]

    for j, img_path in enumerate(category_images):

        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, 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()
```




```
df2.shape
```

```
(806, 2)
```

```
df2.columns
```

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

```
df2.shape
```

```
(806, 2)
```

```
df2.columns
```

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

```
df2.duplicated().sum()
```

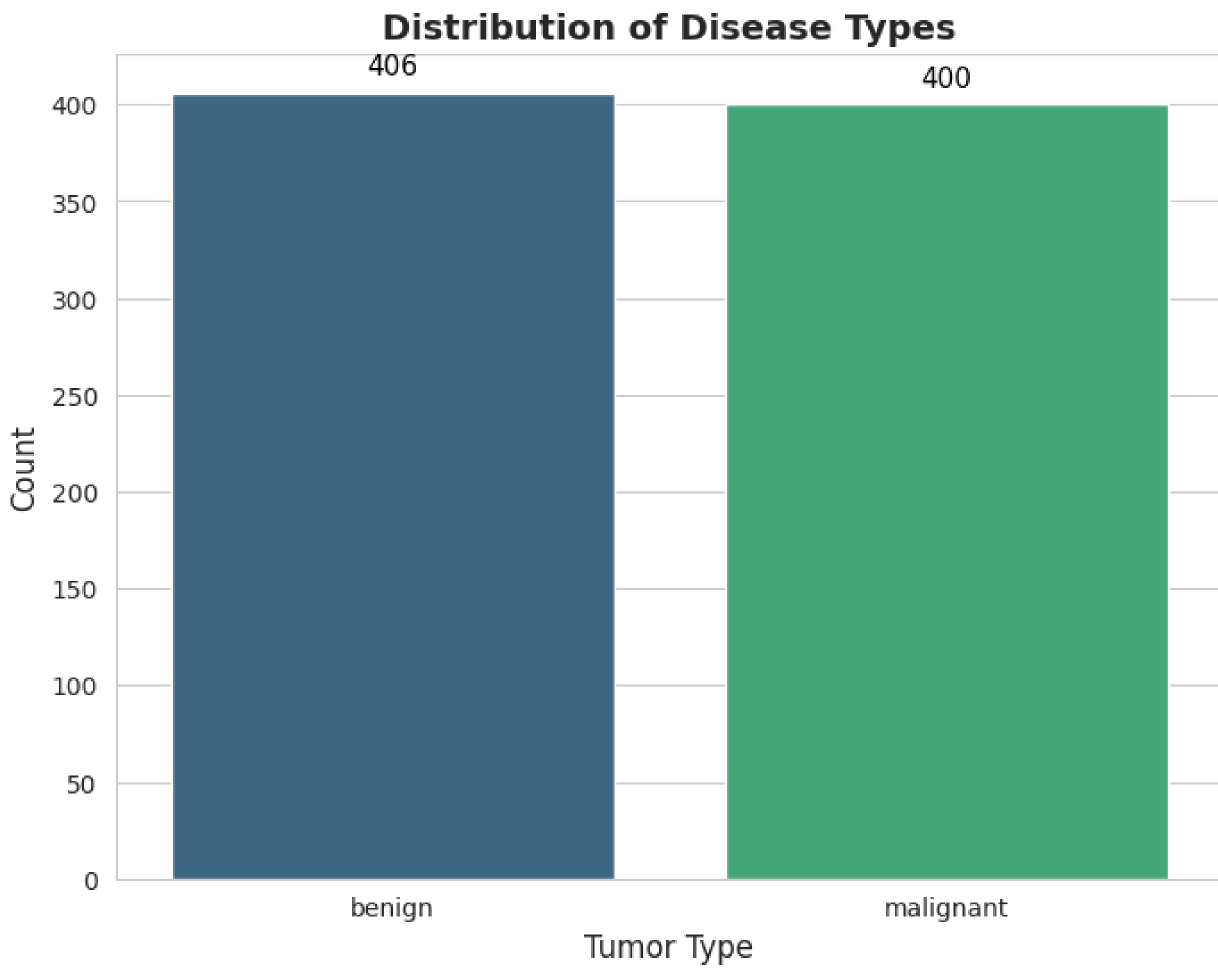
```
0
```

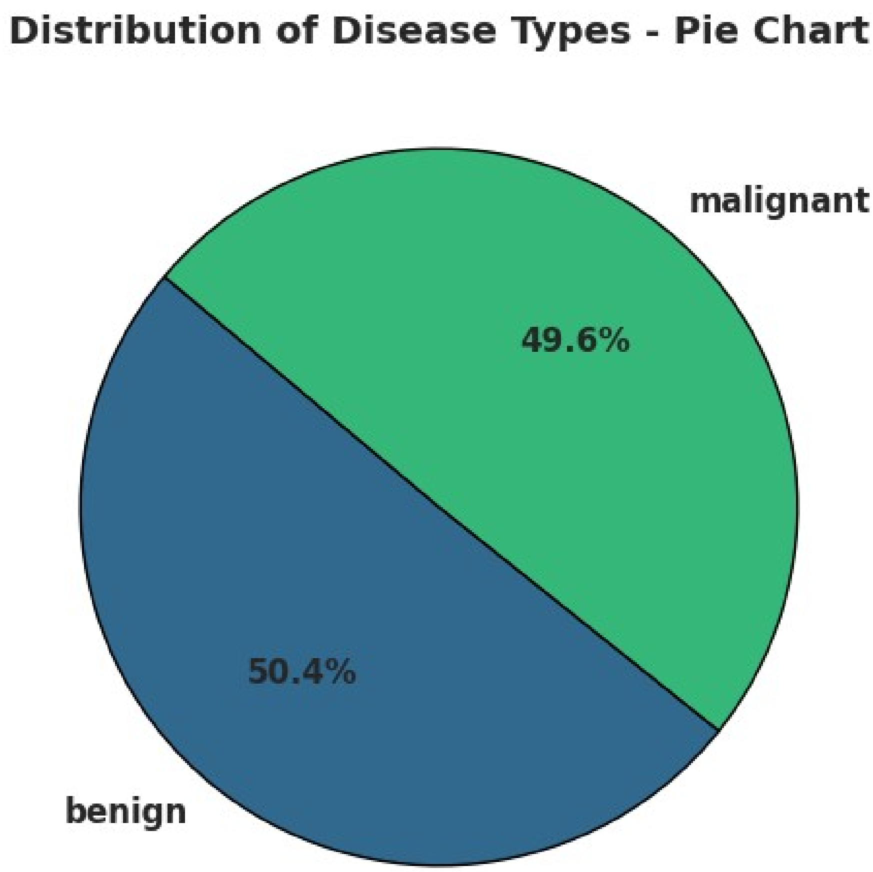
```
df2.isnull().sum()
```

```
image_path    0
label         0
dtype: int64
```

```
df2['label'].unique()
```

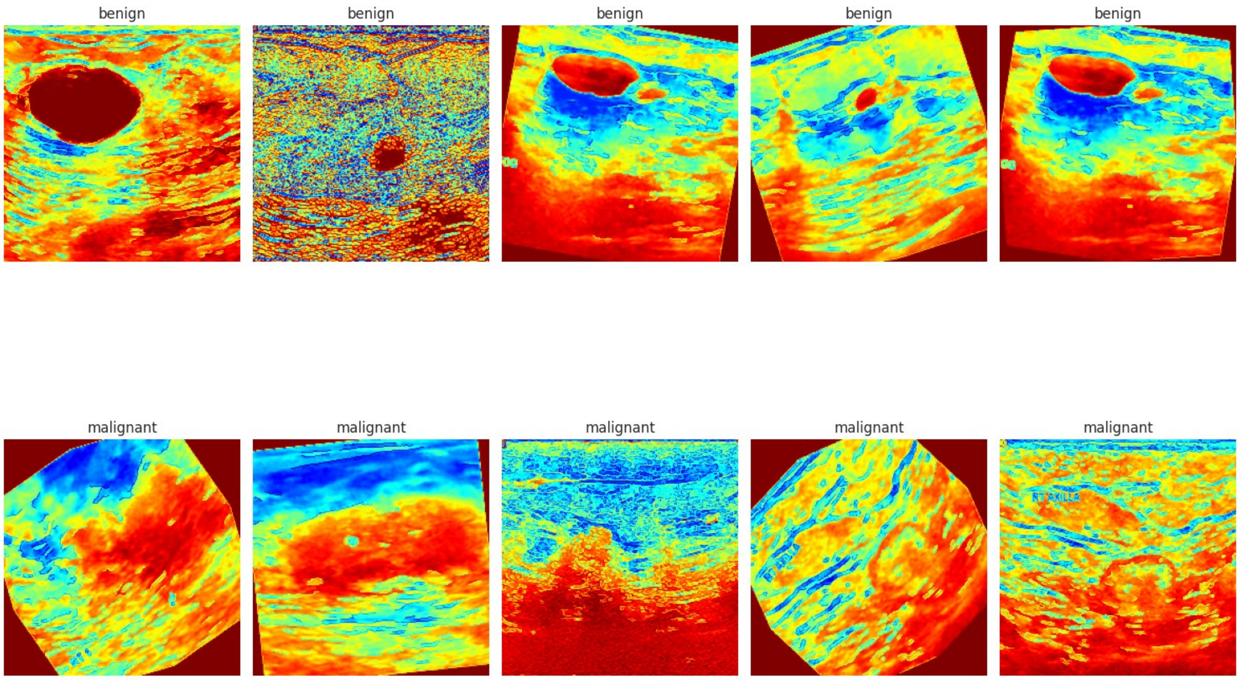
```
array(['benign', 'malignant'], dtype=object)

df2['label'].value_counts()

label
benign       406
malignant    400
Name: count, dtype: int64

import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(8, 6))
sns.countplot(data=df2, x="label", 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 = df2["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 Disease Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution of Disease Types
406
400
400
350
300
250
200
150
100
50
0
Count
benign
malignant
Tumor Type




```
import cv2

num_images = 5

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

for i, category in enumerate(categories):
    category_images = df2[df2['label'] == category]
['image_path'].iloc[:num_images]

    for j, img_path in enumerate(category_images):

        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, 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()
```




```
import os
import cv2
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Concatenate, Input, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
```

```
2025-06-14 07:47:43.450345: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1749887263.973447      35 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1749887264.117622      35 cuda_blas.cc:1418] Unable to
```

```
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

```
def standardize_labels(labels):
    label_map = {
        "Normal": "benign",
        "Malignant": "malignant",
        "benign": "benign",
        "malignant": "malignant"
    }
    return [label_map[label] for label in labels]

def preprocess_image(img_path, target_size=(224, 224)):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, target_size)
    img = img / 255.0
    return img

def load_images(df, modality_name):
    images = []
    labels = []
    for _, row in df.iterrows():
        img = preprocess_image(row['image_path'])
        images.append(img)
        labels.append(row['label'])
    labels = standardize_labels(labels)
    return np.array(images), np.array(labels)

xray_images, xray_labels = load_images(df, "Chest X-ray")
histo_images, histo_labels = load_images(df1, "Histopathological")
ultra_images, ultra_labels = load_images(df2, "Ultrasound")

le = LabelEncoder()
all_labels = np.concatenate([xray_labels, histo_labels, ultra_labels])
le.fit(all_labels)
xray_labels = le.transform(xray_labels)
histo_labels = le.transform(histo_labels)
ultra_labels = le.transform(ultra_labels)

print("Label classes:", le.classes_)
```

```
Label classes: ['benign' 'malignant']
```

```
base_model = ResNet50(weights='imagenet', include_top=False,
pooling='avg')

def extract_features(images, model):
    features = model.predict(images, batch_size=32, verbose=1)
    return features
```

```
xray_features = extract_features(xray_images, base_model)
histo_features = extract_features(histo_images, base_model)
ultra_features = extract_features(ultra_images, base_model)

print("X-ray features shape:", xray_features.shape)
print("Histopathological features shape:", histo_features.shape)
print("Ultrasound features shape:", ultra_features.shape)
```

```
I0000 00:00:1749887554.878941      35 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB
memory:  -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0,
compute capability: 7.5
I0000 00:00:1749887554.879812      35 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB
memory:  -> device: 1, name: Tesla T4, pci bus id: 0000:00:05.0,
compute capability: 7.5

Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/
resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1749887564.015999     158 service.cc:148] XLA service
0x7f5e2814c450 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
I0000 00:00:1749887564.017397     158 service.cc:156]   StreamExecutor
device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1749887564.017416     158 service.cc:156]   StreamExecutor
device (1): Tesla T4, Compute Capability 7.5
I0000 00:00:1749887564.757248     158 cuda_dnn.cc:529] Loaded cuDNN
version 90300

 2/32 ━━━━━━━━━━━━━━━━━━━━ 2s 82ms/step

I0000 00:00:1749887570.005888     158 device_compiler.h:188] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

32/32 ━━━━━━━━━━━━━━━━━━━━ 17s 263ms/step
39/39 ━━━━━━━━━━━━━━━━━━━━ 7s 167ms/step
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 179ms/step
X-ray features shape: (1000, 2048)
Histopathological features shape: (1246, 2048)
Ultrasound features shape: (806, 2048)
```

```
num_samples = min(len(xray_features), len(histo_features),
len(ultra_features))
xray_features = xray_features[:num_samples]
histo_features = histo_features[:num_samples]
```

```
ultra_features = ultra_features[:num_samples]
labels = xray_labels[:num_samples]

combined_features = np.concatenate([xray_features, histo_features,
ultra_features], axis=1)
print("Combined features shape:", combined_features.shape)

X_train, X_test, y_train, y_test = train_test_split(
    combined_features, labels, test_size=0.2, random_state=42,
stratify=labels
)
```

```
Combined features shape: (806, 6144)
```

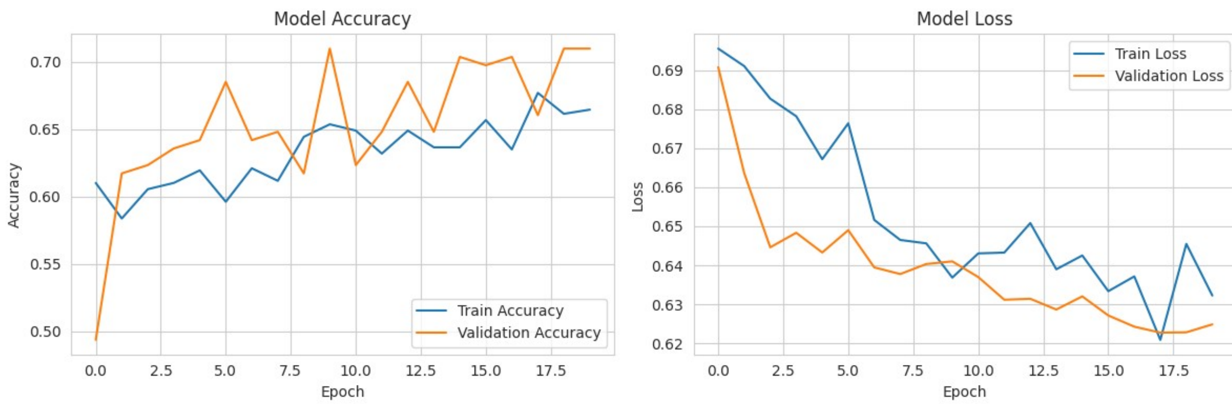
```
input_shape = combined_features.shape[1]
input_layer = Input(shape=(input_shape,))
x = Dense(512, activation='relu')(input_layer)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
output_layer = Dense(1, activation='sigmoid')(x)

model = Model(inputs=input_layer, outputs=output_layer)

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

history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=20,
    batch_size=32,
    verbose=1
)
```

```
Epoch 1/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6462 - loss:
0.6674 - val_accuracy: 0.4938 - val_loss: 0.6909
Epoch 2/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5393 - loss:
0.7028 - val_accuracy: 0.6173 - val_loss: 0.6636
Epoch 3/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5969 - loss:
0.6988 - val_accuracy: 0.6235 - val_loss: 0.6446
Epoch 4/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6175 - loss:
0.6792 - val_accuracy: 0.6358 - val_loss: 0.6483
Epoch 5/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5960 - loss:
0.6866 - val_accuracy: 0.6420 - val_loss: 0.6433
Epoch 6/20
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6151 - loss: 
0.6658 - val_accuracy: 0.6852 - val_loss: 0.6490
Epoch 7/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6094 - loss: 
0.6581 - val_accuracy: 0.6420 - val_loss: 0.6394
Epoch 8/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6179 - loss: 
0.6493 - val_accuracy: 0.6481 - val_loss: 0.6377
Epoch 9/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6408 - loss: 
0.6410 - val_accuracy: 0.6173 - val_loss: 0.6403
Epoch 10/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6776 - loss: 
0.6234 - val_accuracy: 0.7099 - val_loss: 0.6410
Epoch 11/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6653 - loss: 
0.6362 - val_accuracy: 0.6235 - val_loss: 0.6370
Epoch 12/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6342 - loss: 
0.6525 - val_accuracy: 0.6481 - val_loss: 0.6311
Epoch 13/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6367 - loss: 
0.6588 - val_accuracy: 0.6852 - val_loss: 0.6314
Epoch 14/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6487 - loss: 
0.6346 - val_accuracy: 0.6481 - val_loss: 0.6286
Epoch 15/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6513 - loss: 
0.6224 - val_accuracy: 0.7037 - val_loss: 0.6320
Epoch 16/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6686 - loss: 
0.6266 - val_accuracy: 0.6975 - val_loss: 0.6271
Epoch 17/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6299 - loss: 
0.6333 - val_accuracy: 0.7037 - val_loss: 0.6242
Epoch 18/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6377 - loss: 
0.6393 - val_accuracy: 0.6605 - val_loss: 0.6227
Epoch 19/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6402 - loss: 
0.6583 - val_accuracy: 0.7099 - val_loss: 0.6228
Epoch 20/20
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6757 - loss: 
0.6226 - val_accuracy: 0.7099 - val_loss: 0.6248

test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy:.4f}")

y_pred = (model.predict(X_test) > 0.5).astype(int)
print("\nClassification Report:")
```

```python
print(classification_report(y_test, y_pred, target_names=le.classes_))

cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

```
Test Accuracy: 0.7099
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step

Classification Report:
              precision    recall  f1-score   support

      benign       0.71      0.40      0.52        62
   malignant       0.71      0.90      0.79       100

    accuracy                           0.71       162
   macro avg       0.71      0.65      0.65       162
weighted avg       0.71      0.71      0.69       162


Confusion Matrix:
[[25 37]
 [10 90]]
```



Paper Implementation

```
import os
import cv2
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score, classification_report,
confusion_matrix, roc_curve
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet import ResNet152
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Concatenate, Input,
Dropout, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter

def standardize_labels(labels):
    label_map = {
        "Normal": "benign",
        "Malignant": "malignant",
        "benign": "benign",
        "malignant": "malignant"
    }
    return [label_map.get(label, label) for label in labels]

def normalize_grayscale(img):
    x_min, x_max = img.min(), img.max()
    if x_max == x_min:
        return img
    return (img - x_min) / (x_max - x_min)
```

```python
def preprocess_image(img_path, target_size=(224, 224)):
    try:
        img = cv2.imread(img_path)
        if img is None:
            raise ValueError(f"Failed to load image: {img_path}")
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, target_size)
        img = normalize_grayscale(img)
        return img
    except Exception as e:
        print(e)
        return None

def load_images(df, modality_name):
    images = []
    labels = []
    skipped_paths = []

    for i, row in df.iterrows():
        img = preprocess_image(row['image_path'])
        if img is not None:
            images.append(img)
            labels.append(row['label'])
        else:
            skipped_paths.append(row['image_path'])

    if skipped_paths:
        print(f"Skipped {len(skipped_paths)} images in
{modality_name}: {skipped_paths[:5]}...")

    labels = standardize_labels(labels)

    return np.array(images), np.array(labels), skipped_paths

xray_images, xray_labels, xray_skipped = load_images(df, "Chest X-
ray")
histo_images, histo_labels, histo_skipped = load_images(df1,
"Histopathological")
ultra_images, ultra_labels, ultra_skipped = load_images(df2,
"Ultrasound")

le = LabelEncoder()
all_labels = np.concatenate([xray_labels, histo_labels, ultra_labels])
le.fit(all_labels)
xray_labels = le.transform(xray_labels)
histo_labels = le.transform(histo_labels)
ultra_labels = le.transform(ultra_labels)

num_samples = min(len(xray_images), len(histo_images),
len(ultra_images))
```

```
xray_images = xray_images[:num_samples]
xray_labels = xray_labels[:num_samples]
histo_images = histo_images[:num_samples]
histo_labels = histo_labels[:num_samples]
ultra_images = ultra_images[:num_samples]
ultra_labels = ultra_labels[:num_samples]

print("Label classes:", le.classes_)
print("Aligned dataset size:", num_samples)
print("X-ray images shape:", xray_images.shape)
print("X-ray labels distribution:",
pd.Series(le.inverse_transform(xray_labels)).value_counts())
print("Histopathological images shape:", histo_images.shape)
print("Histopathological labels distribution:",
pd.Series(le.inverse_transform(histo_labels)).value_counts())
print("Ultrasound images shape:", ultra_images.shape)
print("Ultrasound labels distribution:",
pd.Series(le.inverse_transform(ultra_labels)).value_counts())
```

```
Label classes: ['benign' 'malignant']
Aligned dataset size: 806
X-ray images shape: (806, 224, 224, 3)
X-ray labels distribution: malignant    500
benign       306
Name: count, dtype: int64
Histopathological images shape: (806, 224, 224, 3)
Histopathological labels distribution: benign       623
malignant    183
Name: count, dtype: int64
Ultrasound images shape: (806, 224, 224, 3)
Ultrasound labels distribution: benign       406
malignant    400
Name: count, dtype: int64
```

```
def augment_images(images, labels, target_class, target_count=1000):
    datagen = ImageDataGenerator(
        horizontal_flip=True,
        vertical_flip=True,
        rotation_range=20,
        zoom_range=0.2,
        fill_mode='nearest'
    )
    malignant_indices = np.where(labels == target_class)[0]
    if len(malignant_indices) == 0:
        print(f"No samples found for target class {target_class}.
Skipping augmentation.")
        return images, labels

    malignant_images = images[malignant_indices]
    current_count = len(malignant_indices)
```

```
    augment_needed = target_count - current_count
    if augment_needed <= 0:
        print(f"Enough samples ({current_count}) for target class
{target_class}. No augmentation needed.")
        return images, labels

    augmented_images = []
    augmented_labels = []
    for _ in range(augment_needed // current_count + 1):
        for img in malignant_images:
            img = img.reshape((1,) + img.shape)
            for batch in datagen.flow(img, batch_size=1):
                augmented_images.append(batch[0])
                augmented_labels.append(target_class)
                if len(augmented_images) >= augment_needed:
                    break
        if len(augmented_images) >= augment_needed:
            break

    augmented_images = np.array(augmented_images[:augment_needed])
    augmented_labels = np.array(augmented_labels[:augment_needed])
    return np.concatenate([images, augmented_images], axis=0),
np.concatenate([labels, augmented_labels], axis=0)

malignant_class = le.transform(['malignant'])[0]
xray_images, xray_labels = augment_images(xray_images, xray_labels,
target_class=malignant_class)
histo_images, histo_labels = augment_images(histo_images,
histo_labels, target_class=malignant_class)
ultra_images, ultra_labels = augment_images(ultra_images,
ultra_labels, target_class=malignant_class)

num_samples = min(len(xray_images), len(histo_images),
len(ultra_images))
xray_images = xray_images[:num_samples]
xray_labels = xray_labels[:num_samples]
histo_images = histo_images[:num_samples]
histo_labels = histo_labels[:num_samples]
ultra_images = ultra_images[:num_samples]
ultra_labels = ultra_labels[:num_samples]

print("Dataset size after augmentation:", num_samples)
print("Label distribution after augmentation:",
pd.Series(le.inverse_transform(xray_labels)).value_counts())
```

```
Dataset size after augmentation: 1306
Label distribution after augmentation: malignant    1000
benign        306
Name: count, dtype: int64
```

```python
def build_feature_extractor():
    base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
    x = GlobalAveragePooling2D()(base_model.output)
    model = Model(inputs=base_model.input, outputs=x)
    return model

def extract_features(images, model):
    features = model.predict(images, batch_size=16, verbose=1)
    return features

xray_extractor = build_feature_extractor()
histo_extractor = build_feature_extractor()
ultra_extractor = build_feature_extractor()

xray_features = extract_features(xray_images, xray_extractor)
histo_features = extract_features(histo_images, histo_extractor)
ultra_features = extract_features(ultra_images, ultra_extractor)

print("X-ray features shape:", xray_features.shape)
print("Histopathological features shape:", histo_features.shape)
print("Ultrasound features shape:", ultra_features.shape)
```

```
82/82 ━━━━━━━━━━━━━━━━━━━━ 16s 122ms/step
82/82 ━━━━━━━━━━━━━━━━━━━━ 13s 106ms/step
82/82 ━━━━━━━━━━━━━━━━━━━━ 13s 105ms/step
X-ray features shape: (1306, 2048)
Histopathological features shape: (1306, 2048)
Ultrasound features shape: (1306, 2048)
```

```python
input_shape = xray_features.shape[1]
xray_input = Input(shape=(input_shape,))
histo_input = Input(shape=(input_shape,))
ultra_input = Input(shape=(input_shape,))

concatenated = Concatenate()([xray_input, histo_input, ultra_input])
x = Dense(512, activation='relu')(concatenated)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=[xray_input, histo_input, ultra_input],
outputs=output)

model.compile(optimizer=Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])

X_train_xray, X_test_xray, X_train_histo, X_test_histo, X_train_ultra,
X_test_ultra, y_train, y_test = train_test_split(
    xray_features, histo_features, ultra_features, xray_labels,
```

```
test_size=0.2, random_state=42, stratify=xray_labels
)

history = model.fit(
    [X_train_xray, X_train_histo, X_train_ultra], y_train,
    validation_data=([X_test_xray, X_test_histo, X_test_ultra],
y_test),
    epochs=200,
    batch_size=16,
    verbose=1
)
```

```
Epoch 1/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7438 - loss:
0.5229 - val_accuracy: 0.7710 - val_loss: 0.4720
Epoch 2/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7707 - loss:
0.4740 - val_accuracy: 0.7786 - val_loss: 0.4673
Epoch 3/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7716 - loss:
0.4776 - val_accuracy: 0.7786 - val_loss: 0.4563
Epoch 4/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7562 - loss:
0.4927 - val_accuracy: 0.7748 - val_loss: 0.4536
Epoch 5/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7569 - loss:
0.4880 - val_accuracy: 0.8092 - val_loss: 0.4553
Epoch 6/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7608 - loss:
0.4831 - val_accuracy: 0.7939 - val_loss: 0.4652
Epoch 7/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7545 - loss:
0.4907 - val_accuracy: 0.7748 - val_loss: 0.4474
Epoch 8/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7622 - loss:
0.4879 - val_accuracy: 0.7824 - val_loss: 0.4442
Epoch 9/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7459 - loss:
0.4827 - val_accuracy: 0.7977 - val_loss: 0.4344
Epoch 10/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7694 - loss:
0.4618 - val_accuracy: 0.7977 - val_loss: 0.4321
Epoch 11/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7583 - loss:
0.4555 - val_accuracy: 0.8015 - val_loss: 0.4326
Epoch 12/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7766 - loss:
0.4533 - val_accuracy: 0.7863 - val_loss: 0.4334
Epoch 13/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7325 - loss:
```

```
0.4891 - val_accuracy: 0.7786 - val_loss: 0.4552
Epoch 14/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7691 - loss:
0.4529 - val_accuracy: 0.8015 - val_loss: 0.4267
Epoch 15/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7871 - loss:
0.4154 - val_accuracy: 0.7939 - val_loss: 0.4308
Epoch 16/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7706 - loss:
0.4388 - val_accuracy: 0.8015 - val_loss: 0.4171
Epoch 17/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8029 - loss:
0.4042 - val_accuracy: 0.7901 - val_loss: 0.4333
Epoch 18/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7761 - loss:
0.4493 - val_accuracy: 0.7977 - val_loss: 0.4156
Epoch 19/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7868 - loss:
0.4155 - val_accuracy: 0.7901 - val_loss: 0.4195
Epoch 20/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7850 - loss:
0.4245 - val_accuracy: 0.7939 - val_loss: 0.4164
Epoch 21/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7700 - loss:
0.4511 - val_accuracy: 0.7977 - val_loss: 0.4111
Epoch 22/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7623 - loss:
0.4311 - val_accuracy: 0.7977 - val_loss: 0.4238
Epoch 23/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7917 - loss:
0.4200 - val_accuracy: 0.8015 - val_loss: 0.4099
Epoch 24/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7756 - loss:
0.4188 - val_accuracy: 0.8015 - val_loss: 0.4046
Epoch 25/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7666 - loss:
0.4446 - val_accuracy: 0.8168 - val_loss: 0.4096
Epoch 26/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7754 - loss:
0.4259 - val_accuracy: 0.8053 - val_loss: 0.3991
Epoch 27/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7993 - loss:
0.4053 - val_accuracy: 0.8206 - val_loss: 0.3962
Epoch 28/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7955 - loss:
0.4065 - val_accuracy: 0.8092 - val_loss: 0.3983
Epoch 29/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7768 - loss:
0.4420 - val_accuracy: 0.8015 - val_loss: 0.3925
```

```
Epoch 30/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8086 - loss:
0.3904 - val_accuracy: 0.8092 - val_loss: 0.3984
Epoch 31/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7952 - loss:
0.4014 - val_accuracy: 0.8130 - val_loss: 0.4019
Epoch 32/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7726 - loss:
0.4157 - val_accuracy: 0.8053 - val_loss: 0.3891
Epoch 33/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7623 - loss:
0.4358 - val_accuracy: 0.7901 - val_loss: 0.4140
Epoch 34/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8125 - loss:
0.3936 - val_accuracy: 0.8092 - val_loss: 0.3869
Epoch 35/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8000 - loss:
0.3910 - val_accuracy: 0.8244 - val_loss: 0.3851
Epoch 36/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8209 - loss:
0.3752 - val_accuracy: 0.8206 - val_loss: 0.3802
Epoch 37/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8067 - loss:
0.3839 - val_accuracy: 0.8130 - val_loss: 0.3891
Epoch 38/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8024 - loss:
0.3893 - val_accuracy: 0.8168 - val_loss: 0.3792
Epoch 39/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8174 - loss:
0.3769 - val_accuracy: 0.8244 - val_loss: 0.3815
Epoch 40/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7948 - loss:
0.3897 - val_accuracy: 0.8397 - val_loss: 0.3798
Epoch 41/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7716 - loss:
0.4074 - val_accuracy: 0.8206 - val_loss: 0.3776
Epoch 42/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8386 - loss:
0.3482 - val_accuracy: 0.8206 - val_loss: 0.3785
Epoch 43/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8020 - loss:
0.3805 - val_accuracy: 0.8168 - val_loss: 0.3773
Epoch 44/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7942 - loss:
0.3805 - val_accuracy: 0.8435 - val_loss: 0.3680
Epoch 45/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8019 - loss:
0.3986 - val_accuracy: 0.8168 - val_loss: 0.3717
Epoch 46/200
```

```
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8100 - loss:
0.3757 - val_accuracy: 0.8397 - val_loss: 0.3729
Epoch 47/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8356 - loss:
0.3361 - val_accuracy: 0.8244 - val_loss: 0.3671
Epoch 48/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8236 - loss:
0.3796 - val_accuracy: 0.8435 - val_loss: 0.3652
Epoch 49/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8025 - loss:
0.3872 - val_accuracy: 0.8244 - val_loss: 0.3698
Epoch 50/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8058 - loss:
0.3683 - val_accuracy: 0.8282 - val_loss: 0.3765
Epoch 51/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8075 - loss:
0.3961 - val_accuracy: 0.8244 - val_loss: 0.3742
Epoch 52/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8168 - loss:
0.3639 - val_accuracy: 0.8473 - val_loss: 0.3591
Epoch 53/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8091 - loss:
0.3614 - val_accuracy: 0.8130 - val_loss: 0.3633
Epoch 54/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8016 - loss:
0.3633 - val_accuracy: 0.8359 - val_loss: 0.3637
Epoch 55/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8321 - loss:
0.3509 - val_accuracy: 0.8435 - val_loss: 0.3581
Epoch 56/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8256 - loss:
0.3620 - val_accuracy: 0.8397 - val_loss: 0.3600
Epoch 57/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8225 - loss:
0.3640 - val_accuracy: 0.8206 - val_loss: 0.3679
Epoch 58/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7847 - loss:
0.4004 - val_accuracy: 0.8435 - val_loss: 0.3555
Epoch 59/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8245 - loss:
0.3650 - val_accuracy: 0.8282 - val_loss: 0.3729
Epoch 60/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8114 - loss:
0.3668 - val_accuracy: 0.8206 - val_loss: 0.3843
Epoch 61/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8163 - loss:
0.3848 - val_accuracy: 0.8473 - val_loss: 0.3595
Epoch 62/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8234 - loss:
```

```
0.3561 - val_accuracy: 0.8130 - val_loss: 0.3662
Epoch 63/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8320 - loss:
0.3357 - val_accuracy: 0.8435 - val_loss: 0.3519
Epoch 64/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8164 - loss:
0.3453 - val_accuracy: 0.8435 - val_loss: 0.3457
Epoch 65/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8358 - loss:
0.3359 - val_accuracy: 0.8130 - val_loss: 0.3615
Epoch 66/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8182 - loss:
0.3565 - val_accuracy: 0.8473 - val_loss: 0.3432
Epoch 67/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8197 - loss:
0.3481 - val_accuracy: 0.8282 - val_loss: 0.3544
Epoch 68/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8333 - loss:
0.3580 - val_accuracy: 0.8015 - val_loss: 0.4004
Epoch 69/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8286 - loss:
0.3469 - val_accuracy: 0.8244 - val_loss: 0.3679
Epoch 70/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8338 - loss:
0.3622 - val_accuracy: 0.8511 - val_loss: 0.3418
Epoch 71/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8347 - loss:
0.3429 - val_accuracy: 0.8511 - val_loss: 0.3472
Epoch 72/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8296 - loss:
0.3411 - val_accuracy: 0.8206 - val_loss: 0.3533
Epoch 73/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8091 - loss:
0.3485 - val_accuracy: 0.8550 - val_loss: 0.3369
Epoch 74/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8438 - loss:
0.3340 - val_accuracy: 0.8511 - val_loss: 0.3424
Epoch 75/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8505 - loss:
0.3235 - val_accuracy: 0.8473 - val_loss: 0.3486
Epoch 76/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8166 - loss:
0.3451 - val_accuracy: 0.8168 - val_loss: 0.3543
Epoch 77/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8609 - loss:
0.3129 - val_accuracy: 0.8550 - val_loss: 0.3415
Epoch 78/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8144 - loss:
0.3388 - val_accuracy: 0.8092 - val_loss: 0.3634
```

```
Epoch 79/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8527 - loss:
0.3169 - val_accuracy: 0.8550 - val_loss: 0.3427
Epoch 80/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8571 - loss:
0.3219 - val_accuracy: 0.8435 - val_loss: 0.3342
Epoch 81/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8227 - loss:
0.3395 - val_accuracy: 0.8511 - val_loss: 0.3365
Epoch 82/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8338 - loss:
0.3367 - val_accuracy: 0.8511 - val_loss: 0.3352
Epoch 83/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8117 - loss:
0.3451 - val_accuracy: 0.7939 - val_loss: 0.3788
Epoch 84/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8362 - loss:
0.3296 - val_accuracy: 0.8473 - val_loss: 0.3291
Epoch 85/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8242 - loss:
0.3354 - val_accuracy: 0.8550 - val_loss: 0.3357
Epoch 86/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8345 - loss:
0.3241 - val_accuracy: 0.8473 - val_loss: 0.3317
Epoch 87/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8462 - loss:
0.3143 - val_accuracy: 0.8397 - val_loss: 0.3298
Epoch 88/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8387 - loss:
0.3234 - val_accuracy: 0.8359 - val_loss: 0.3640
Epoch 89/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8319 - loss:
0.3229 - val_accuracy: 0.8244 - val_loss: 0.3599
Epoch 90/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8370 - loss:
0.3295 - val_accuracy: 0.8511 - val_loss: 0.3245
Epoch 91/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8506 - loss:
0.3123 - val_accuracy: 0.8473 - val_loss: 0.3315
Epoch 92/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8348 - loss:
0.3112 - val_accuracy: 0.8473 - val_loss: 0.3301
Epoch 93/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8468 - loss:
0.3156 - val_accuracy: 0.8511 - val_loss: 0.3220
Epoch 94/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8501 - loss:
0.3274 - val_accuracy: 0.8435 - val_loss: 0.3491
Epoch 95/200
```

```
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8278 - loss:
0.3439 - val_accuracy: 0.8168 - val_loss: 0.3749
Epoch 96/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8442 - loss:
0.3167 - val_accuracy: 0.8473 - val_loss: 0.3359
Epoch 97/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8517 - loss:
0.3092 - val_accuracy: 0.8397 - val_loss: 0.3238
Epoch 98/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8536 - loss:
0.3248 - val_accuracy: 0.8588 - val_loss: 0.3300
Epoch 99/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8349 - loss:
0.3188 - val_accuracy: 0.8550 - val_loss: 0.3205
Epoch 100/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8524 - loss:
0.3250 - val_accuracy: 0.8511 - val_loss: 0.3271
Epoch 101/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8546 - loss:
0.3054 - val_accuracy: 0.8511 - val_loss: 0.3311
Epoch 102/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8555 - loss:
0.3268 - val_accuracy: 0.8397 - val_loss: 0.3521
Epoch 103/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8312 - loss:
0.3375 - val_accuracy: 0.8168 - val_loss: 0.3793
Epoch 104/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8310 - loss:
0.3224 - val_accuracy: 0.8511 - val_loss: 0.3291
Epoch 105/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8475 - loss:
0.3130 - val_accuracy: 0.8664 - val_loss: 0.3181
Epoch 106/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8535 - loss:
0.3128 - val_accuracy: 0.8588 - val_loss: 0.3148
Epoch 107/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8566 - loss:
0.3125 - val_accuracy: 0.8397 - val_loss: 0.3325
Epoch 108/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8503 - loss:
0.3114 - val_accuracy: 0.8626 - val_loss: 0.3178
Epoch 109/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8316 - loss:
0.3333 - val_accuracy: 0.8664 - val_loss: 0.3208
Epoch 110/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8751 - loss:
0.2742 - val_accuracy: 0.8664 - val_loss: 0.3185
Epoch 111/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8475 - loss:
```

```
0.3153 - val_accuracy: 0.8550 - val_loss: 0.3229
Epoch 112/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8270 - loss:
0.3207 - val_accuracy: 0.8626 - val_loss: 0.3163
Epoch 113/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8599 - loss:
0.2893 - val_accuracy: 0.8664 - val_loss: 0.3183
Epoch 114/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8556 - loss:
0.2844 - val_accuracy: 0.8740 - val_loss: 0.3098
Epoch 115/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8590 - loss:
0.2898 - val_accuracy: 0.8321 - val_loss: 0.3497
Epoch 116/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8506 - loss:
0.3143 - val_accuracy: 0.8588 - val_loss: 0.3127
Epoch 117/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8632 - loss:
0.2823 - val_accuracy: 0.8282 - val_loss: 0.3801
Epoch 118/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8590 - loss:
0.3077 - val_accuracy: 0.8702 - val_loss: 0.3099
Epoch 119/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8528 - loss:
0.3083 - val_accuracy: 0.8550 - val_loss: 0.3442
Epoch 120/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8464 - loss:
0.3151 - val_accuracy: 0.8779 - val_loss: 0.3140
Epoch 121/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8602 - loss:
0.2828 - val_accuracy: 0.8435 - val_loss: 0.3433
Epoch 122/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8457 - loss:
0.3142 - val_accuracy: 0.8626 - val_loss: 0.3177
Epoch 123/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8567 - loss:
0.2858 - val_accuracy: 0.8588 - val_loss: 0.3322
Epoch 124/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8584 - loss:
0.3073 - val_accuracy: 0.8550 - val_loss: 0.3203
Epoch 125/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8575 - loss:
0.2808 - val_accuracy: 0.8321 - val_loss: 0.3322
Epoch 126/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8707 - loss:
0.2843 - val_accuracy: 0.8664 - val_loss: 0.3197
Epoch 127/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8703 - loss:
0.2961 - val_accuracy: 0.8626 - val_loss: 0.3061
```

```
Epoch 128/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8451 - loss:
0.3134 - val_accuracy: 0.8664 - val_loss: 0.3327
Epoch 129/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8677 - loss:
0.2796 - val_accuracy: 0.8550 - val_loss: 0.3212
Epoch 130/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8434 - loss:
0.3111 - val_accuracy: 0.8664 - val_loss: 0.3032
Epoch 131/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8549 - loss:
0.2958 - val_accuracy: 0.8664 - val_loss: 0.3051
Epoch 132/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8469 - loss:
0.3070 - val_accuracy: 0.8626 - val_loss: 0.3136
Epoch 133/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8626 - loss:
0.2914 - val_accuracy: 0.8626 - val_loss: 0.3173
Epoch 134/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8664 - loss:
0.2795 - val_accuracy: 0.8626 - val_loss: 0.3048
Epoch 135/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8511 - loss:
0.3040 - val_accuracy: 0.8664 - val_loss: 0.3139
Epoch 136/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8471 - loss:
0.3102 - val_accuracy: 0.8779 - val_loss: 0.3096
Epoch 137/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8606 - loss:
0.2922 - val_accuracy: 0.8588 - val_loss: 0.3193
Epoch 138/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8726 - loss:
0.2797 - val_accuracy: 0.8664 - val_loss: 0.3164
Epoch 139/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8660 - loss:
0.2939 - val_accuracy: 0.8359 - val_loss: 0.3258
Epoch 140/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8322 - loss:
0.3210 - val_accuracy: 0.8321 - val_loss: 0.3438
Epoch 141/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8469 - loss:
0.3167 - val_accuracy: 0.8588 - val_loss: 0.3103
Epoch 142/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8571 - loss:
0.2987 - val_accuracy: 0.8664 - val_loss: 0.3020
Epoch 143/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8631 - loss:
0.2968 - val_accuracy: 0.8588 - val_loss: 0.3286
Epoch 144/200
```

```
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8532 - loss:
0.3278 - val_accuracy: 0.8359 - val_loss: 0.3501
Epoch 145/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8355 - loss:
0.3199 - val_accuracy: 0.8321 - val_loss: 0.3484
Epoch 146/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8835 - loss:
0.2605 - val_accuracy: 0.8740 - val_loss: 0.3032
Epoch 147/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8869 - loss:
0.2745 - val_accuracy: 0.8168 - val_loss: 0.3489
Epoch 148/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8612 - loss:
0.3118 - val_accuracy: 0.8779 - val_loss: 0.3133
Epoch 149/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8460 - loss:
0.3108 - val_accuracy: 0.8473 - val_loss: 0.3184
Epoch 150/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8609 - loss:
0.2955 - val_accuracy: 0.8206 - val_loss: 0.3386
Epoch 151/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8546 - loss:
0.3083 - val_accuracy: 0.8626 - val_loss: 0.3051
Epoch 152/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8612 - loss:
0.2926 - val_accuracy: 0.8779 - val_loss: 0.3037
Epoch 153/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8604 - loss:
0.2852 - val_accuracy: 0.8626 - val_loss: 0.3096
Epoch 154/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8592 - loss:
0.3015 - val_accuracy: 0.8626 - val_loss: 0.3088
Epoch 155/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8823 - loss:
0.2850 - val_accuracy: 0.8664 - val_loss: 0.3106
Epoch 156/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8588 - loss:
0.3019 - val_accuracy: 0.8626 - val_loss: 0.3119
Epoch 157/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8496 - loss:
0.3091 - val_accuracy: 0.8702 - val_loss: 0.3072
Epoch 158/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8801 - loss:
0.2820 - val_accuracy: 0.8473 - val_loss: 0.3340
Epoch 159/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8656 - loss:
0.2753 - val_accuracy: 0.8740 - val_loss: 0.3145
Epoch 160/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8744 - loss:
```

```
0.2739 - val_accuracy: 0.8702 - val_loss: 0.3190
Epoch 161/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8727 - loss:
0.2852 - val_accuracy: 0.8740 - val_loss: 0.3187
Epoch 162/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8709 - loss:
0.2759 - val_accuracy: 0.8740 - val_loss: 0.2996
Epoch 163/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8653 - loss:
0.2815 - val_accuracy: 0.8740 - val_loss: 0.2954
Epoch 164/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8658 - loss:
0.2944 - val_accuracy: 0.8626 - val_loss: 0.3130
Epoch 165/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8749 - loss:
0.2866 - val_accuracy: 0.8588 - val_loss: 0.3007
Epoch 166/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8449 - loss:
0.3489 - val_accuracy: 0.8740 - val_loss: 0.3037
Epoch 167/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8825 - loss:
0.2613 - val_accuracy: 0.8779 - val_loss: 0.3070
Epoch 168/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8647 - loss:
0.2857 - val_accuracy: 0.8626 - val_loss: 0.3205
Epoch 169/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8745 - loss:
0.2616 - val_accuracy: 0.8588 - val_loss: 0.3251
Epoch 170/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8641 - loss:
0.2918 - val_accuracy: 0.8588 - val_loss: 0.3332
Epoch 171/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8852 - loss:
0.2488 - val_accuracy: 0.8702 - val_loss: 0.3043
Epoch 172/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8818 - loss:
0.2777 - val_accuracy: 0.8779 - val_loss: 0.2923
Epoch 173/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8406 - loss:
0.3124 - val_accuracy: 0.8664 - val_loss: 0.3057
Epoch 174/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8839 - loss:
0.2628 - val_accuracy: 0.8702 - val_loss: 0.3153
Epoch 175/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8440 - loss:
0.3079 - val_accuracy: 0.8511 - val_loss: 0.3087
Epoch 176/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8612 - loss:
0.2997 - val_accuracy: 0.8740 - val_loss: 0.3014
Epoch 177/200
```

```
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8648 - loss:
0.3000 - val_accuracy: 0.8740 - val_loss: 0.2994
Epoch 178/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8460 - loss:
0.3068 - val_accuracy: 0.8588 - val_loss: 0.3344
Epoch 179/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8834 - loss:
0.2848 - val_accuracy: 0.8779 - val_loss: 0.2961
Epoch 180/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8633 - loss:
0.2978 - val_accuracy: 0.8702 - val_loss: 0.3089
Epoch 181/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8903 - loss:
0.2674 - val_accuracy: 0.8664 - val_loss: 0.3162
Epoch 182/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8738 - loss:
0.2900 - val_accuracy: 0.8626 - val_loss: 0.2987
Epoch 183/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8654 - loss:
0.3021 - val_accuracy: 0.8626 - val_loss: 0.3046
Epoch 184/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8909 - loss:
0.2660 - val_accuracy: 0.8588 - val_loss: 0.3024
Epoch 185/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8662 - loss:
0.2923 - val_accuracy: 0.8282 - val_loss: 0.3869
Epoch 186/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8577 - loss:
0.2917 - val_accuracy: 0.8664 - val_loss: 0.2914
Epoch 187/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8707 - loss:
0.2717 - val_accuracy: 0.8740 - val_loss: 0.3145
Epoch 188/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8738 - loss:
0.2705 - val_accuracy: 0.8282 - val_loss: 0.3421
Epoch 189/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8800 - loss:
0.2610 - val_accuracy: 0.8702 - val_loss: 0.3241
Epoch 190/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8632 - loss:
0.2747 - val_accuracy: 0.8740 - val_loss: 0.3006
Epoch 191/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8504 - loss:
0.3085 - val_accuracy: 0.8511 - val_loss: 0.3113
Epoch 192/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8645 - loss:
0.2911 - val_accuracy: 0.8702 - val_loss: 0.3130
Epoch 193/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8711 - loss:
```

```
0.2712 - val_accuracy: 0.8588 - val_loss: 0.3192
Epoch 194/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8634 - loss:
0.2988 - val_accuracy: 0.8664 - val_loss: 0.3047
Epoch 195/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8802 - loss:
0.2785 - val_accuracy: 0.8702 - val_loss: 0.3004
Epoch 196/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8761 - loss:
0.2698 - val_accuracy: 0.8511 - val_loss: 0.3387
Epoch 197/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8638 - loss:
0.2945 - val_accuracy: 0.8588 - val_loss: 0.3207
Epoch 198/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8656 - loss:
0.2800 - val_accuracy: 0.8702 - val_loss: 0.3055
Epoch 199/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8639 - loss:
0.2938 - val_accuracy: 0.8626 - val_loss: 0.3051
Epoch 200/200
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8676 - loss:
0.2768 - val_accuracy: 0.8740 - val_loss: 0.3030
```

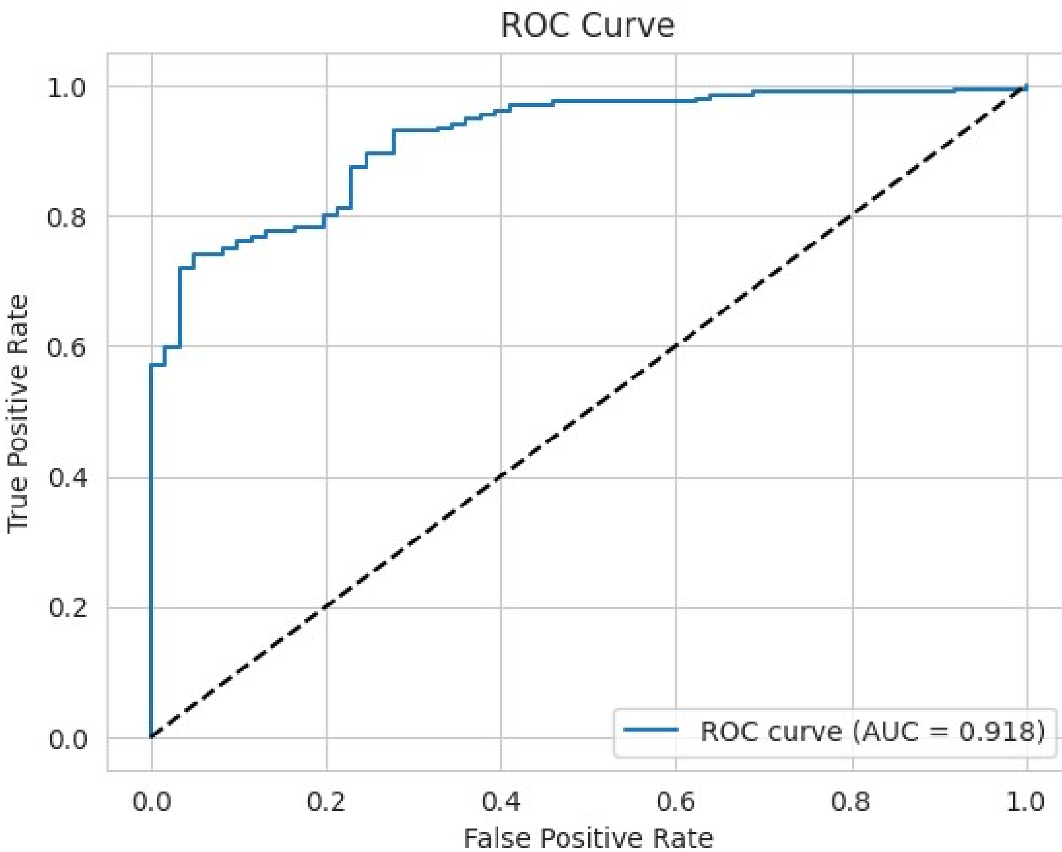
```
y_pred_prob = model.predict([X_test_xray, X_test_histo, X_test_ultra])
y_pred = (y_pred_prob > 0.5).astype(int)
```

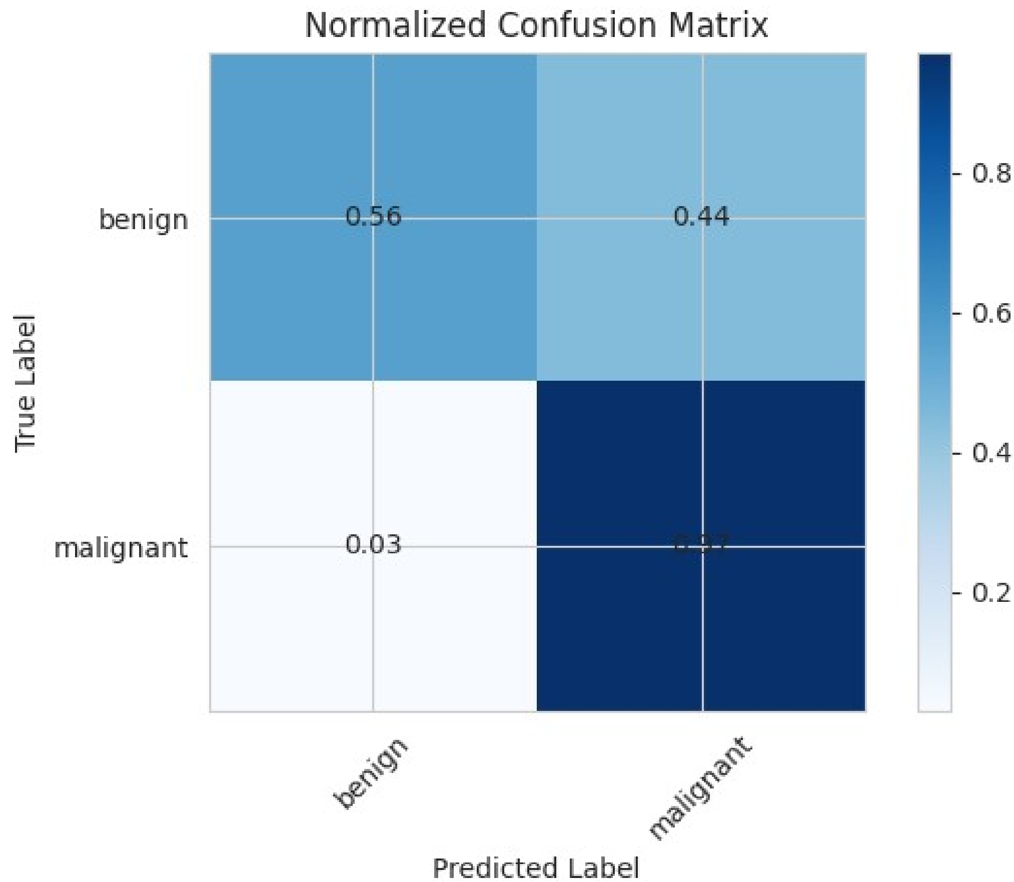
```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

```
auc = roc_auc_score(y_test, y_pred_prob)
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
accuracy = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn) > 0
else 0
precision = tp / (tp + fp) if (tp + fp) > 0 else 0

print(f"AUC: {auc:.3f}")
print(f"Sensitivity: {sensitivity:.3f}")
print(f"Specificity: {specificity:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Accuracy: {accuracy:.3f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=le.classes_))
print("\nConfusion Matrix:")
print(cm)

fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.3f})')
```

```python
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()

cm_norm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
plt.figure()
plt.imshow(cm_norm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Normalized Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(le.classes_))
plt.xticks(tick_marks, le.classes_, rotation=45)
plt.yticks(tick_marks, le.classes_)
for i, j in np.ndindex(cm_norm.shape):
    plt.text(j, i, f'{cm_norm[i, j]:.2f}', ha='center', va='center')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.tight_layout()
plt.show()
```

```
AUC: 0.918
Sensitivity: 0.970
Specificity: 0.557
Precision: 0.878
Accuracy: 0.874

Classification Report:
              precision    recall  f1-score   support

      benign       0.85      0.56      0.67        61
   malignant       0.88      0.97      0.92       201

    accuracy                           0.87       262
   macro avg       0.86      0.76      0.80       262
weighted avg       0.87      0.87      0.86       262


Confusion Matrix:
[[ 34  27]
 [  6 195]]
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.918)


Normalized Confusion Matrix
True Label
benign
malignant
0.56
0.44
0.03
0.8
0.6
0.4
0.2
benign
malignant
Predicted Label