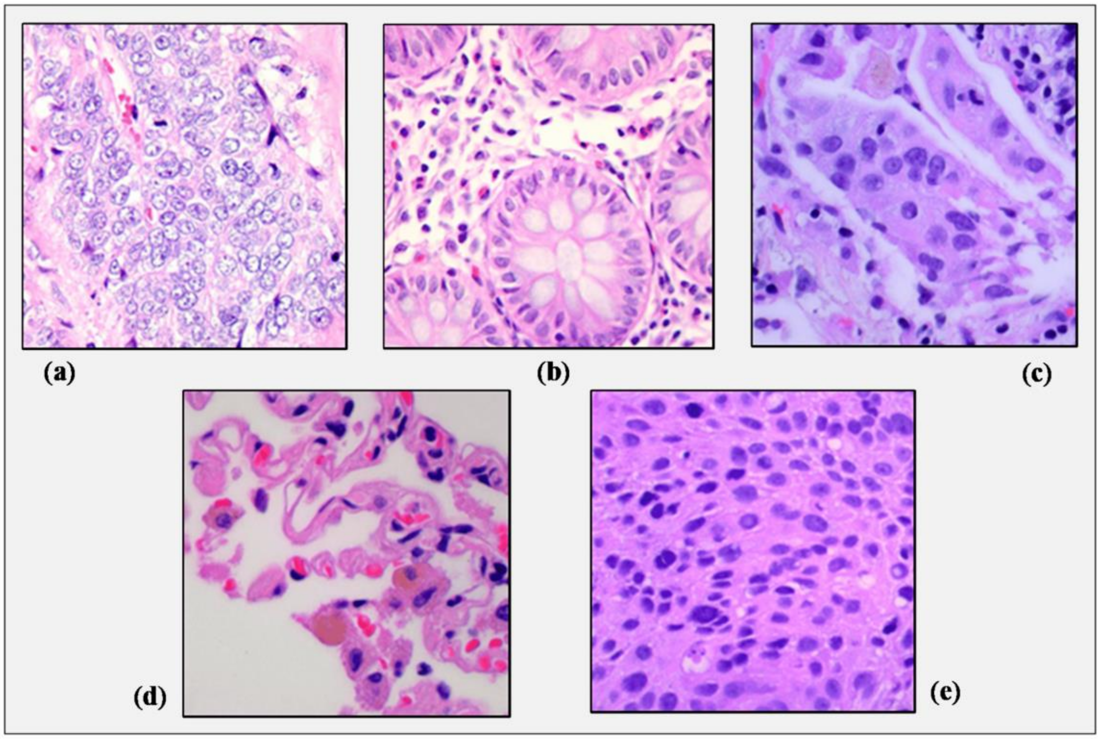
MRANet with SE Block for Lung Colon Cancer Classification



```
import numpy as np
import pandas as pd

base_path = "/kaggle/input/lung-and-colon-cancer-histopathological-images/lung_colon_image_set/colon_image_sets"
categories = ["colon_aca","colon_n"]

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)


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

df.head()
```

```
                                          image_path      label
0  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_aca
1  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_aca
2  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_aca
3  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_aca
4  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_aca
```

```
df.tail()
```

```
                                             image_path    label
9995  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_n
9996  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_n
9997  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_n
9998  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_n
9999  /kaggle/input/lung-and-colon-cancer-histopatho...  colon_n
```

```
df.shape
```

```
(10000, 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: 10000 entries, 0 to 9999
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  10000 non-null  object
 1   label       10000 non-null  object
dtypes: object(2)
memory usage: 156.4+ KB
```

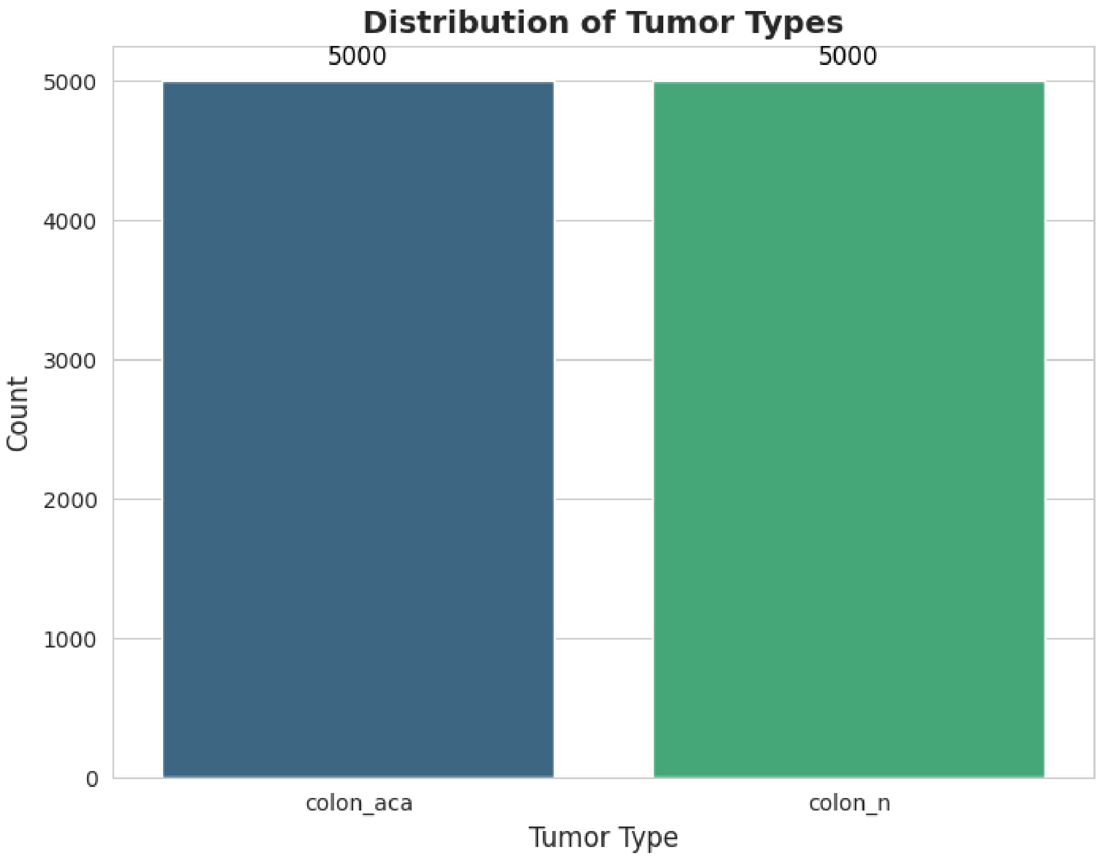
```
df['label'].unique()
```

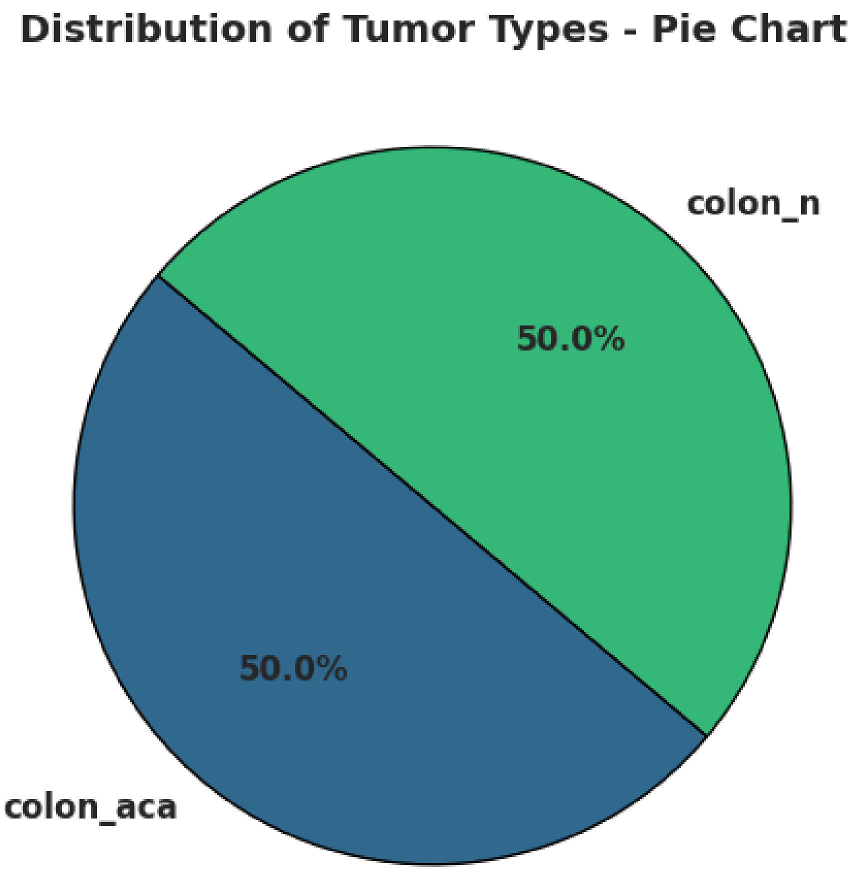
```
array(['colon_aca', 'colon_n'], dtype=object)
```

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

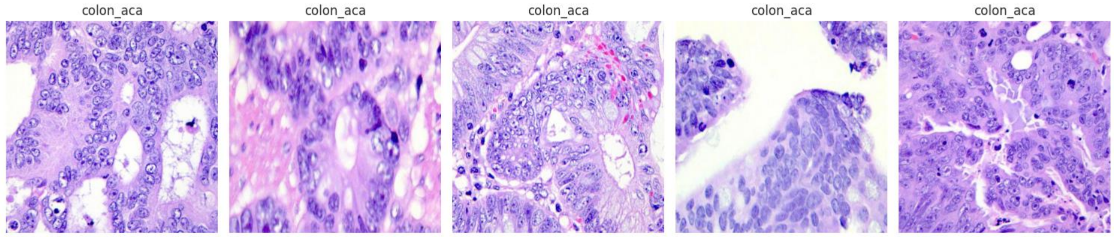
```
label
colon_aca    5000
colon_n      5000
Name: count, dtype: int64
```

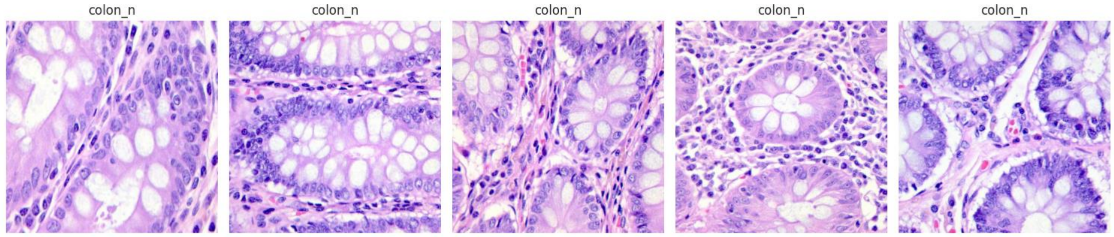
```
import seaborn as sns
import matplotlib.pyplot as plt

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 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
5000
5000
5000
4000
3000
2000
1000
0
Count
colon_aca
colon_n
Tumor Type



```
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()
```





```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

df['category_encoded'] = label_encoder.fit_transform(df['label'])

df = df[['image_path', 'category_encoded']]

from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(df[['image_path']],
df['category_encoded'])

df_resampled = pd.DataFrame(X_resampled, columns=['image_path'])
df_resampled['category_encoded'] = y_resampled

print("\nClass distribution after oversampling:")
print(df_resampled['category_encoded'].value_counts())
```

```
Class distribution after oversampling:
category_encoded
0    5000
1    5000
Name: count, dtype: int64
```

```
df_resampled
```

```
                                             image_path  category_encoded
0     /kaggle/input/lung-and-colon-cancer-histopatho...                 0
1     /kaggle/input/lung-and-colon-cancer-histopatho...                 0
2     /kaggle/input/lung-and-colon-cancer-histopatho...                 0
3     /kaggle/input/lung-and-colon-cancer-histopatho...                 0
4     /kaggle/input/lung-and-colon-cancer-histopatho...                 0
...                                                 ...               ...
9995  /kaggle/input/lung-and-colon-cancer-histopatho...                 1
9996  /kaggle/input/lung-and-colon-cancer-histopatho...                 1
9997  /kaggle/input/lung-and-colon-cancer-histopatho...                 1
9998  /kaggle/input/lung-and-colon-cancer-histopatho...                 1
9999  /kaggle/input/lung-and-colon-cancer-histopatho...                 1

[10000 rows x 2 columns]
```

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

```python
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
```

```python
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']
)

batch_size = 16
img_size = (256, 256)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

tr_gen = ImageDataGenerator(rescale=1./255)
ts_gen = ImageDataGenerator(rescale=1./255)

train_gen_new = tr_gen.flow_from_dataframe(
    train_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='binary',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size
)

valid_gen_new = ts_gen.flow_from_dataframe(
    valid_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='binary',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size
)

test_gen_new = ts_gen.flow_from_dataframe(
    test_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='binary',
    color_mode='rgb',
    shuffle=False,
    batch_size=batch_size
)
```

```
Found 8000 validated image filenames belonging to 2 classes.
Found 1000 validated image filenames belonging to 2 classes.
Found 1000 validated image filenames belonging to 2 classes.
```

```
from tensorflow.keras.layers import Input, Conv2D, GlobalAveragePooling2D,
Dense, Multiply, Reshape, BatchNormalization, Activation
```

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  2
```

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print("GPU is set for TensorFlow")
    except RuntimeError as e:
        print(e)
```

```
GPU is set for TensorFlow
```

```
from tensorflow.keras.layers import Add
```

```
from tensorflow.keras.layers import Add, GlobalAveragePooling2D, Dense,
BatchNormalization, Reshape, Multiply, Input, UpSampling2D
```

```
from tensorflow.keras.layers import Add, GlobalAveragePooling2D, Dense,
BatchNormalization, Reshape, Multiply, Input, Conv2D, UpSampling2D
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import backend as K

def attention_block(inputs, filters):
    x = GlobalAveragePooling2D()(inputs)
    x = Dense(filters // 16, activation='relu')(x)
    x = Dense(filters, activation='sigmoid')(x)
    x = Reshape((1, 1, filters))(x)
    return Multiply()([inputs, x])

def build_mranet(input_shape=(256, 256, 3), num_classes=2):
    base_model = ResNet50(weights='imagenet', include_top=False,
input_tensor=Input(shape=input_shape))

    conv4_block6_out = base_model.get_layer('conv4_block6_out').output
    conv5_block3_out = base_model.get_layer('conv5_block3_out').output

    attn_block1 = attention_block(conv4_block6_out, filters=1024)
    attn_block2 = attention_block(conv5_block3_out, filters=2048)

    attn_block2 = Conv2D(1024, (1, 1), padding='same',
activation='relu')(attn_block2)
```

```
    attn_block2 = UpSampling2D((2, 2))(attn_block2)

    attn_block1_shape = K.int_shape(attn_block1)
    attn_block2_shape = K.int_shape(attn_block2)
    print("Shape of attn_block1:", attn_block1_shape)
    print("Shape of attn_block2:", attn_block2_shape)

    merged_attention = Add()([attn_block1, attn_block2])

    x = GlobalAveragePooling2D()(merged_attention)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = BatchNormalization()(x)
    x = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=base_model.input, outputs=x)

    for layer in base_model.layers:
        layer.trainable = False

    return model

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

```
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
Shape of attn_block1: (None, 16, 16, 1024)
Shape of attn_block2: (None, 16, 16, 1024)
```

```
from PIL import Image

def my_image_check(generator):
    while True:
        x, y = next(generator)
        new_x = []
        new_y = []
        for i in range(x.shape[0]):
            try:
                img = Image.fromarray((x[i] * 255).astype(np.uint8))
                img.verify()
                new_x.append(x[i])
                new_y.append(y[i])
            except Exception as e:
                print(f"Error loading image:
{generator.dataframe['image_path'].iloc[generator.index + i] if
(generator.index + i) < len(generator.dataframe) else 'Path information not
available'}")
                print(f"Error: {e}")
        yield np.array(new_x), np.array(new_y)

tr_gen = ImageDataGenerator(rescale=1./255)
ts_gen = ImageDataGenerator(rescale=1./255)

train_generator = tr_gen.flow_from_dataframe(
    train_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='binary',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    stratify=df_resampled['category_encoded']
)
train_gen_new = my_image_check(train_generator)

valid_generator = ts_gen.flow_from_dataframe(
    valid_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='binary',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    stratify=df_resampled['category_encoded']
)

valid_gen_new = my_image_check(valid_generator)

test_generator = ts_gen.flow_from_dataframe(
    test_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='binary',
    color_mode='rgb',
    shuffle=False,
    batch_size=batch_size,
    stratify=df_resampled['category_encoded']
)

test_gen_new = my_image_check(test_generator)
```

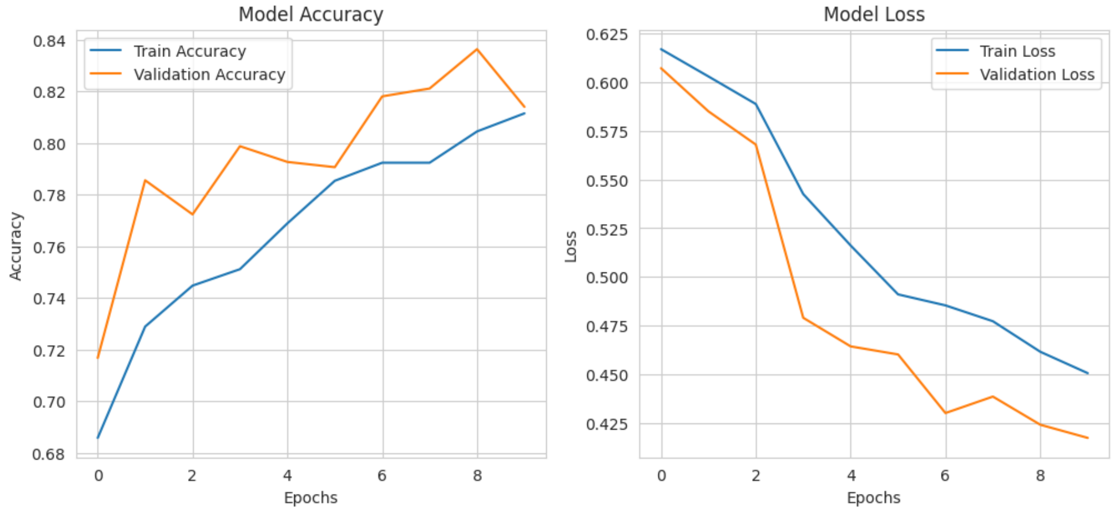
```python
steps_per_epoch = len(train_df_new) // batch_size
validation_steps = len(valid_df_new) // batch_size
```

```
Found 8000 validated image filenames belonging to 2 classes.
Found 1000 validated image filenames belonging to 2 classes.
Found 1000 validated image filenames belonging to 2 classes.
```

```python
history = mranet_model.fit(
    train_gen_new,
    epochs=10,
    validation_data=valid_gen_new,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
)
```

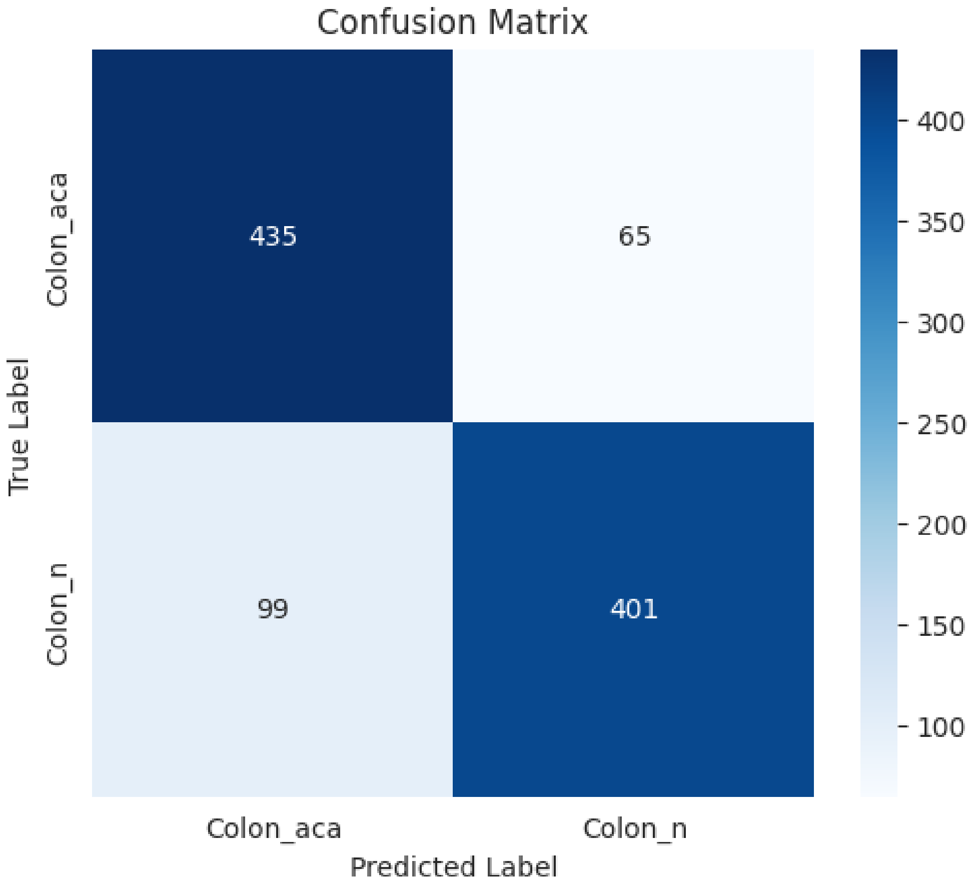
```
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 98s 163ms/step - accuracy: 0.6369 - loss: 0.6445
- val_accuracy: 0.7167 - val_loss: 0.6070
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 51s 101ms/step - accuracy: 0.7280 - loss: 0.6018
- val_accuracy: 0.7856 - val_loss: 0.5849
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 49s 98ms/step - accuracy: 0.7563 - loss: 0.5835
- val_accuracy: 0.7724 - val_loss: 0.5678
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 46s 92ms/step - accuracy: 0.7453 - loss: 0.5601
- val_accuracy: 0.7988 - val_loss: 0.4791
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 46s 92ms/step - accuracy: 0.7732 - loss: 0.5173
- val_accuracy: 0.7927 - val_loss: 0.4644
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 43s 86ms/step - accuracy: 0.7851 - loss: 0.4913
- val_accuracy: 0.7907 - val_loss: 0.4603
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 44s 88ms/step - accuracy: 0.7960 - loss: 0.4876
- val_accuracy: 0.8181 - val_loss: 0.4303
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 45s 89ms/step - accuracy: 0.7832 - loss: 0.4902
- val_accuracy: 0.8211 - val_loss: 0.4387
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 45s 90ms/step - accuracy: 0.8063 - loss: 0.4549
- val_accuracy: 0.8364 - val_loss: 0.4243
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 45s 90ms/step - accuracy: 0.8039 - loss: 0.4662
- val_accuracy: 0.8140 - val_loss: 0.4175
```

```python
import matplotlib.pyplot as plt

def plot_history(history):
    plt.figure(figsize=(12, 5))
    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('Epochs')
    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('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

plot_history(history)
```



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

y_true = test_generator.classes
y_pred = mranet_model.predict(test_generator)
y_pred_classes = np.round(y_pred).astype(int)

cm = confusion_matrix(y_true, y_pred_classes)

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

print("Classification Report:\n", classification_report(y_true,
y_pred_classes))
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 171ms/step
```



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

           0       0.81      0.87      0.84       500
           1       0.86      0.80      0.83       500

    accuracy                           0.84      1000
   macro avg       0.84      0.84      0.84      1000
weighted avg       0.84      0.84      0.84      1000
```

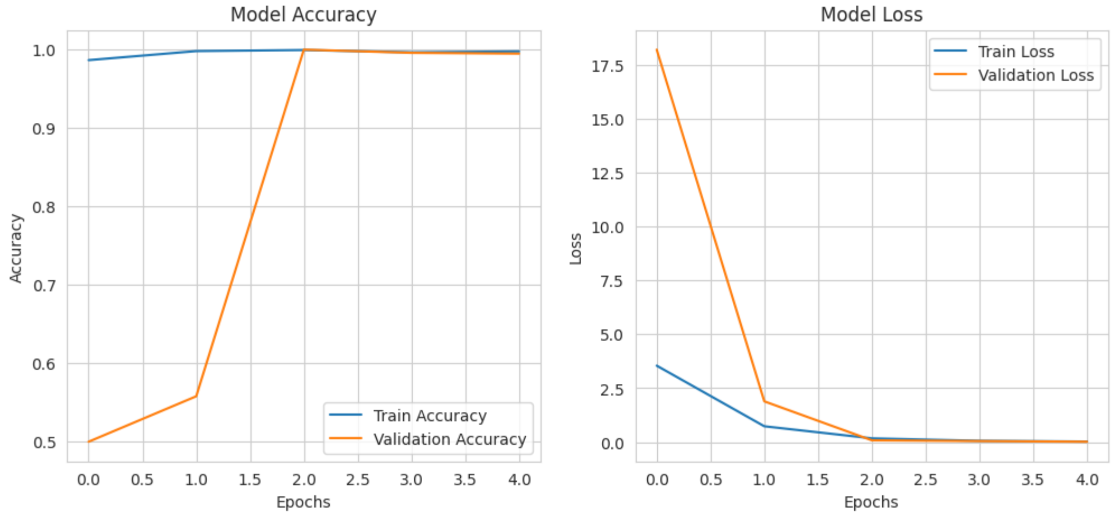
```
import tensorflow as tf
from tensorflow.keras.layers import Add, GlobalAveragePooling2D, Dense,
BatchNormalization, Reshape, Multiply, Input, Conv2D, UpSampling2D, Dropout
```

```python
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ReduceLROnPlateau
import numpy as np

def se_block(input_tensor, ratio=8):
    channels = input_tensor.shape[-1]
    se = GlobalAveragePooling2D()(input_tensor)
    se = Dense(channels // ratio, activation='relu')(se)
    se = Dense(channels, activation='sigmoid')(se)
    se = Reshape((1, 1, channels))(se)
    return Multiply()([input_tensor, se])

def build_mranet(input_shape=(256, 256, 3), num_classes=1):
    base_model = ResNet50(weights='imagenet', include_top=False,
input_tensor=Input(shape=input_shape))

    conv4_block6_out = base_model.get_layer('conv4_block6_out').output
    conv5_block3_out = base_model.get_layer('conv5_block3_out').output

    attn_block1 = se_block(conv4_block6_out, ratio=8)
    attn_block2 = se_block(conv5_block3_out, ratio=8)

    attn_block2 = Conv2D(1024, (3, 3), dilation_rate=(2, 2), padding='same',
activation='relu')(attn_block2)
    attn_block2 = UpSampling2D((2, 2))(attn_block2)

    attn_block1 = Conv2D(1024, (1, 1), padding='same',
activation='relu')(attn_block1)

    merged_attention = Add()([attn_block1, attn_block2])

    x = GlobalAveragePooling2D()(merged_attention)
    x = Dense(512, activation='relu', kernel_regularizer=l2(0.01))(x)
    x = Dropout(0.5)(x)
    x = BatchNormalization()(x)
    x = Dense(num_classes, activation='sigmoid')(x)

    model = Model(inputs=base_model.input, outputs=x)

    for layer in base_model.layers[-10:]:
        layer.trainable = True

    return model

mranet_model = build_mranet()
```

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

lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5)

history = mranet_model.fit(
    train_gen_new,
    epochs=5,
    validation_data=valid_gen_new,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
)
```

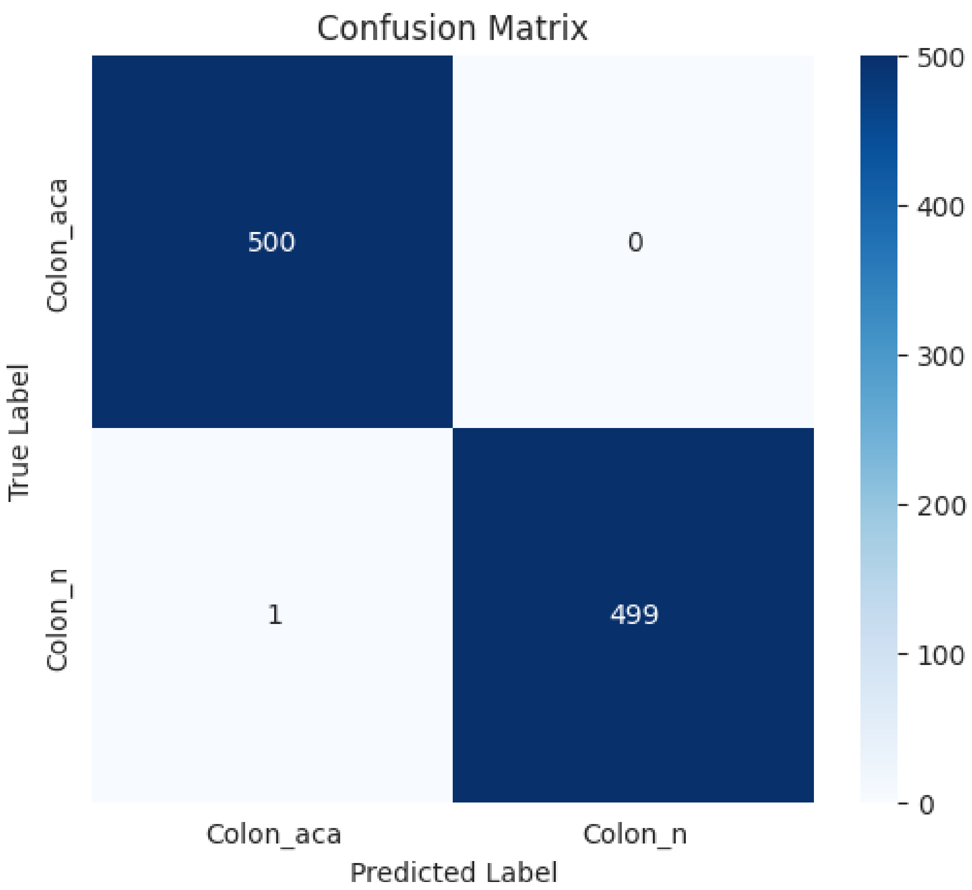
```
Epoch 1/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 191s 255ms/step - accuracy: 0.9715 - loss:
5.0099 - val_accuracy: 0.5000 - val_loss: 18.1901
Epoch 2/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 120s 240ms/step - accuracy: 0.9983 - loss:
1.0274 - val_accuracy: 0.5579 - val_loss: 1.8907
Epoch 3/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 116s 233ms/step - accuracy: 0.9991 - loss:
0.2367 - val_accuracy: 1.0000 - val_loss: 0.0838
Epoch 4/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 115s 229ms/step - accuracy: 0.9990 - loss:
0.0673 - val_accuracy: 0.9959 - val_loss: 0.0498
Epoch 5/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 114s 228ms/step - accuracy: 0.9967 - loss:
0.0344 - val_accuracy: 0.9949 - val_loss: 0.0199
```

```
def plot_history(history):
    plt.figure(figsize=(12, 5))

    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('Epochs')
    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('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

plot_history(history)
```



```
y_true = test_generator.classes
y_pred = mranet_model.predict(test_generator)
y_pred_classes = np.round(y_pred).astype(int)

cm = confusion_matrix(y_true, y_pred_classes)

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

print("Classification Report:\n", classification_report(y_true,
y_pred_classes))
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 132ms/step
```



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

           0       1.00      1.00      1.00       500
           1       1.00      1.00      1.00       500

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000
```