Libraries

```
import os
from os import listdir

import pandas as pd
import numpy as np
from numpy import hstack
import matplotlib.pyplot as plt
from matplotlib.image import imread
import plotly.express as px
import seaborn as sns

import tensorflow as tf
from keras.utils import to_categorical
from keras.preprocessing import image
from keras.layers import Dense, Flatten, Activation, Dropout
from keras import optimizers
```

Raw Data Import

```
from google.colab import drive
drive.mount('/content/drive/')

!ls /content/drive/
```

```
Mounted at /content/drive/
MyDrive
```

Data Preprocessing

```
dicom_info = pd.read_csv('/content/drive/MyDrive/CIND860/csv/dicom_info.csv')
mass_case_test = pd.read_csv('/content/drive/MyDrive/CIND860/csv/mass_case_description_test_set.csv')
mass_case_train = pd.read_csv('/content/drive/MyDrive/CIND860/csv/mass_case_description_train_set.csv')
image_dir = '/content/drive/MyDrive/CIND860/jpeg'


full_mammogram_images = dicom_info[dicom_info.SeriesDescription == 'full mammogram images'].image_path
cropped_images = dicom_info[dicom_info.SeriesDescription == 'cropped images'].image_path
roi_mask_images = dicom_info[dicom_info.SeriesDescription == 'ROI mask images'].image_path

full_mammogram_images = full_mammogram_images.apply(lambda x: x.replace('CBIS-DDSM/jpeg', image_dir))
cropped_images = cropped_images.apply(lambda x: x.replace('CBIS-DDSM/jpeg', image_dir))
roi_mask_images = roi_mask_images.apply(lambda x: x.replace('CBIS-DDSM/jpeg', image_dir))
full_mammogram_images.iloc[0]
```

```
'/content/drive/MyDrive/CIND860/jpeg/1.3.6.1.4.1.9590.100.1.2.248386742010678582309005372213277814849/1-249.jpg'
```

Preparing the image paths for preprocessing

```
full_mammogram_dict = dict()
cropped_dict = dict()
roi_mask_dict = dict()

for dicom in full_mammogram_images:
    key = dicom.split("/")[6]
    full_mammogram_dict[key] = dicom
for dicom in cropped_images:
    key = dicom.split("/")[6]
    cropped_dict[key] = dicom
for dicom in roi_mask_images:
    key = dicom.split("/")[6]
    roi_mask_dict[key] = dicom
next(iter((full_mammogram_dict.items())) )
```

```
('1.3.6.1.4.1.9590.100.1.2.248386742010678582309005372213277814849',
 '/content/drive/MyDrive/CIND860/jpeg/1.3.6.1.4.1.9590.100.1.2.248386742010678582309005372213277814849/1-249.jpg')
```

```
def fix_image_path(dataset):
    for i, img in enumerate(dataset.values):
        img_name = img[11].split("/")[2]
        dataset.iloc[i,11] = full_mammogram_dict[img_name]
        img_name = img[12].split("/")[2]
        dataset.iloc[i,12] = cropped_dict[img_name]
        img_name = img[13].split("/")[2]
        dataset.iloc[i,13] = roi_mask_dict[img_name]


fix_image_path(mass_case_train)
fix_image_path(mass_case_test)
mass_data = mass_case_train.append(mass_case_test)
mass_data.head()
```

```
<ipython-input-7-5d9cb52cd42a>:3: FutureWarning: The frame.append method is deprec
  mass_data = mass_case_train.append(mass_case_test)
```

	patient_id	breast_density	left or right breast	image view	abnormality id	abnormality type	
0	P_00001	3	LEFT	CC	1	mass	ARCHITECT
1	P_00001	3	LEFT	MLO	1	mass	ARCHITECT
2	P_00004	3	LEFT	CC	1	mass	ARCHITECT
3	P_00004	3	LEFT	MLO	1	mass	ARCHITECT
4	P_00004	3	RIGHT	MLO	1	mass	

Data Cleaning

```
dicom_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10237 entries, 0 to 10236
Data columns (total 38 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   file_path                                     10237 non-null  object
 1   image_path                                    10237 non-null  object
 2   AccessionNumber                               0 non-null      float64
 3   BitsAllocated                                 10237 non-null  int64
 4   BitsStored                                    10237 non-null  int64
 5   BodyPartExamined                              10237 non-null  object
 6   Columns                                       10237 non-null  int64
 7   ContentDate                                   10237 non-null  int64
 8   ContentTime                                   10237 non-null  float64
 9   ConversionType                                10237 non-null  object
 10  HighBit                                       10237 non-null  int64
 11  InstanceNumber                                10237 non-null  int64
 12  LargestImagePixelValue                        10237 non-null  int64
 13  Laterality                                    9671 non-null   object
 14  Modality                                      10237 non-null  object
 15  PatientBirthDate                              0 non-null      float64
 16  PatientID                                     10237 non-null  object
 17  PatientName                                   10237 non-null  object
 18  PatientOrientation                            10237 non-null  object
 19  PatientSex                                    0 non-null      float64
 20  PhotometricInterpretation                     10237 non-null  object
 21  PixelRepresentation                           10237 non-null  int64
 22  ReferringPhysicianName                        0 non-null      float64
 23  Rows                                          10237 non-null  int64
 24  SOPClassUID                                   10237 non-null  object
 25  SOPInstanceUID                                10237 non-null  object
 26  SamplesPerPixel                               10237 non-null  int64
 27  SecondaryCaptureDeviceManufacturer            10237 non-null  object
 28  SecondaryCaptureDeviceManufacturerModelName   10237 non-null  object
 29  SeriesDescription                             9671 non-null   object
 30  SeriesInstanceUID                             10237 non-null  object
```

```
 31  SeriesNumber                                10237 non-null  int64
 32  SmallestImagePixelValue                     10237 non-null  int64
 33  SpecificCharacterSet                        10237 non-null  object
 34  StudyDate                                   9671 non-null   float64
 35  StudyID                                     10237 non-null  object
 36  StudyInstanceUID                            10237 non-null  object
 37  StudyTime                                   9671 non-null   float64
dtypes: float64(7), int64(12), object(19)
memory usage: 3.0+ MB
```

```
dicom_info.drop(['PatientBirthDate','AccessionNumber','Columns','ContentDate','ContentTime','PatientSex','PatientBirthDate',
                                            'ReferringPhysicianName','Rows','SOPClassUID','SOPInstanceUID',
                                            'StudyDate','StudyID','StudyInstanceUID','StudyTime','InstanceNumber','SeriesInstanceUID','S
```

```
dicom_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10237 entries, 0 to 10236
Data columns (total 21 columns):
 #   Column                                      Non-Null Count  Dtype
---  ------                                      --------------  -----
 0   file_path                                   10237 non-null  object
 1   image_path                                  10237 non-null  object
 2   BitsAllocated                               10237 non-null  int64
 3   BitsStored                                  10237 non-null  int64
 4   BodyPartExamined                            10237 non-null  object
 5   ConversionType                              10237 non-null  object
 6   HighBit                                     10237 non-null  int64
 7   LargestImagePixelValue                      10237 non-null  int64
 8   Laterality                                  9671 non-null   object
 9   Modality                                    10237 non-null  object
 10  PatientID                                   10237 non-null  object
 11  PatientName                                 10237 non-null  object
 12  PatientOrientation                          10237 non-null  object
 13  PhotometricInterpretation                   10237 non-null  object
 14  PixelRepresentation                         10237 non-null  int64
 15  SamplesPerPixel                             10237 non-null  int64
 16  SecondaryCaptureDeviceManufacturer          10237 non-null  object
 17  SecondaryCaptureDeviceManufacturerModelName  10237 non-null  object
 18  SeriesDescription                           9671 non-null   object
 19  SmallestImagePixelValue                     10237 non-null  int64
 20  SpecificCharacterSet                        10237 non-null  object
dtypes: int64(7), object(14)
memory usage: 1.6+ MB
```

```
dicom_info.isna().sum()
```

```
file_path                                     0
image_path                                    0
BitsAllocated                                 0
BitsStored                                    0
BodyPartExamined                              0
ConversionType                                0
HighBit                                       0
LargestImagePixelValue                        0
Laterality                                  566
Modality                                      0
PatientID                                     0
PatientName                                   0
PatientOrientation                            0
PhotometricInterpretation                     0
PixelRepresentation                           0
SamplesPerPixel                               0
SecondaryCaptureDeviceManufacturer            0
SecondaryCaptureDeviceManufacturerModelName   0
SeriesDescription                           566
SmallestImagePixelValue                       0
SpecificCharacterSet                          0
dtype: int64
```

```
dicom_info['SeriesDescription'].fillna(method = 'bfill', axis = 0, inplace=True)
dicom_info['Laterality'].fillna(method = 'bfill', axis = 0, inplace=True)
```

```
dicom_info.isna().sum()
```

```
file_path                                     0
image_path                                    0
BitsAllocated                                 0
BitsStored                                    0
```

```
BodyPartExamined                          0
ConversionType                            0
HighBit                                   0
LargestImagePixelValue                    0
Laterality                                0
Modality                                  0
PatientID                                 0
PatientName                               0
PatientOrientation                        0
PhotometricInterpretation                 0
PixelRepresentation                       0
SamplesPerPixel                           0
SecondaryCaptureDeviceManufacturer        0
SecondaryCaptureDeviceManufacturerModelName    0
SeriesDescription                         0
SmallestImagePixelValue                   0
SpecificCharacterSet                      0
dtype: int64
```

```
dicom_info['SeriesDescription'].value_counts()
```

```
cropped images           3859
ROI mask images          3340
full mammogram images    3038
Name: SeriesDescription, dtype: int64
```

```
mass_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1696 entries, 0 to 377
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   patient_id               1696 non-null   object
 1   breast_density           1696 non-null   int64 
 2   left or right breast     1696 non-null   object
 3   image view               1696 non-null   object
 4   abnormality id           1696 non-null   int64 
 5   abnormality type         1696 non-null   object
 6   mass shape               1692 non-null   object
 7   mass margins             1636 non-null   object
 8   assessment               1696 non-null   int64 
 9   pathology                1696 non-null   object
 10  subtlety                 1696 non-null   int64 
 11  image file path          1696 non-null   object
 12  cropped image file path  1696 non-null   object
 13  ROI mask file path       1696 non-null   object
dtypes: int64(4), object(10)
memory usage: 198.8+ KB
```

```
mass_data['image view'] = mass_data['image view'].astype('category')
mass_data['mass margins'] = mass_data['mass margins'].astype('category')
mass_data['mass shape'] = mass_data['mass shape'].astype('category')
mass_data['abnormality type'] = mass_data['abnormality type'].astype('category')
mass_data['pathology'] = mass_data['pathology'].astype('category')
mass_data.isna().sum()
```

```
patient_id                  0
breast_density              0
left or right breast        0
image view                  0
abnormality id              0
abnormality type            0
mass shape                  4
mass margins               60
assessment                  0
pathology                   0
subtlety                    0
image file path             0
cropped image file path     0
ROI mask file path          0
dtype: int64
```

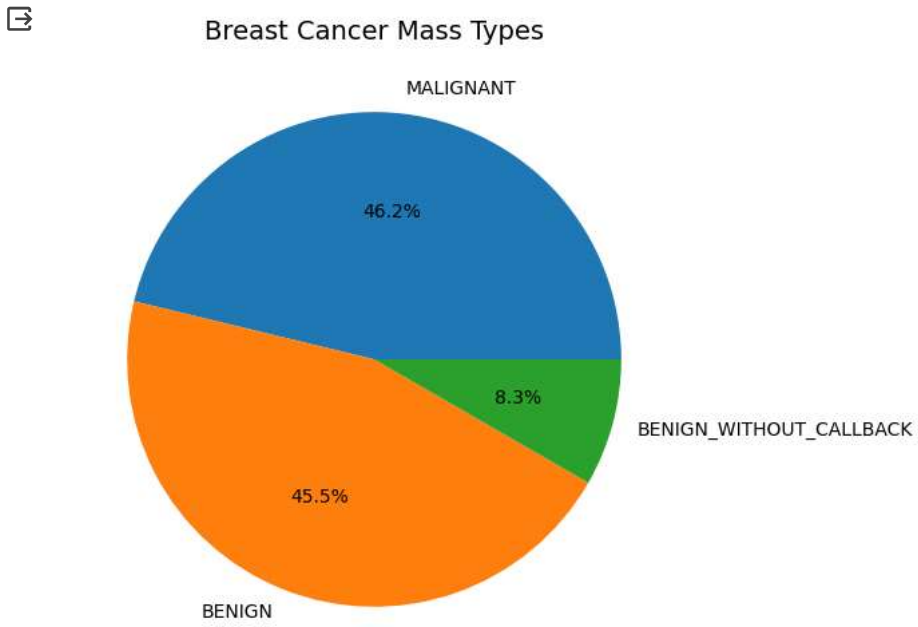
```
mass_data['mass shape'].fillna(method = 'bfill', axis = 0, inplace=True)
mass_data['mass margins'].fillna(method = 'bfill', axis = 0, inplace=True)
```

```
mass_data.isna().sum()
```

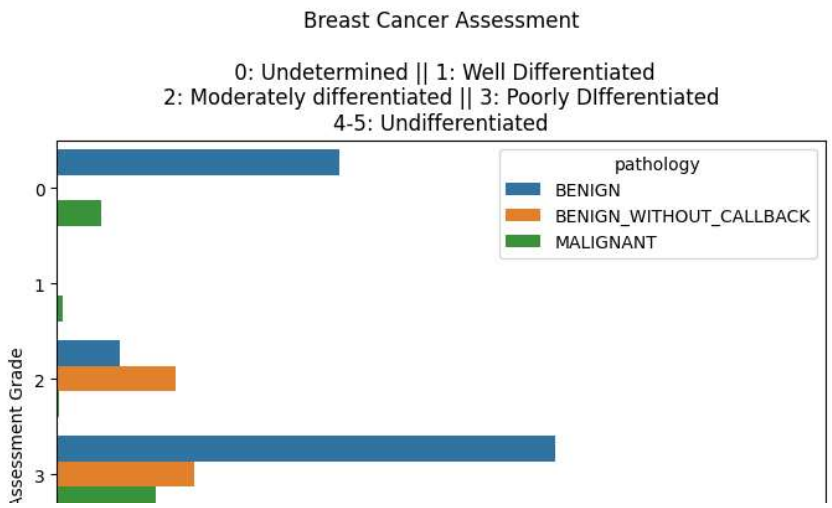
```
patient_id                0
breast_density            0
left or right breast      0
image view                0
abnormality id            0
abnormality type          0
mass shape                0
mass margins              0
assessment                0
pathology                 0
subtlety                  0
image file path           0
cropped image file path   0
ROI mask file path        0
dtype: int64
```

Data Visualization

```
value = mass_data['pathology'].value_counts()
plt.figure(figsize=(8,6))

plt.pie(value, labels=value.index, autopct='%1.1f%%')
plt.title('Breast Cancer Mass Types', fontsize=14)
plt.show()
```

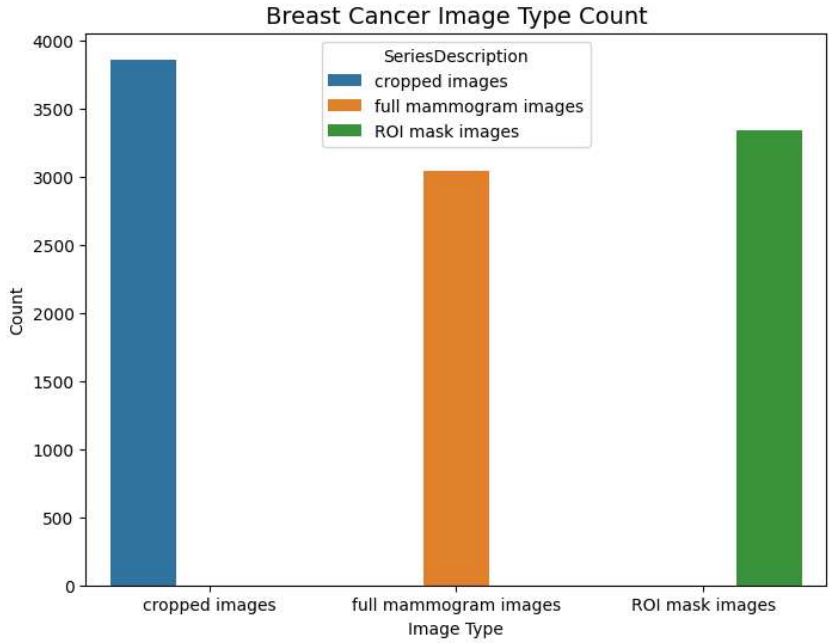



```
plt.figure(figsize=(8,6))
sns.countplot(mass_data, y='assessment', hue='pathology')
plt.title('Breast Cancer Assessment\n\n 0: Undetermined || 1: Well Differentiated\n2: Moderately differentiated || 3: Poorly DIfferentiated\
          fontsize=12)
plt.ylabel('Assessment Grade')
plt.xlabel('Count')
plt.show()
```




```
images = dicom_info['SeriesDescription'].value_counts()
images = images.reset_index()
images = images.rename(columns={'index':'SeriesDescription','SeriesDescription':'counts'})
```

```
images = dicom_info['SeriesDescription'].value_counts()
plt.figure(figsize=(8,6))

sns.countplot(dicom_info, x='SeriesDescription', hue='SeriesDescription', orient='h')
plt.title('Breast Cancer Image Type Count', fontsize=14)
plt.xlabel('Image Type')
plt.ylabel('Count')
plt.show()
```




```
images = dicom_info['SeriesDescription'].value_counts()
plt.figure(figsize=(8,6))

plt.pie(images, labels=images.index, autopct='%1.1f%%')
plt.title('Breast Cancer Image Types', fontsize=14)
plt.show()
```

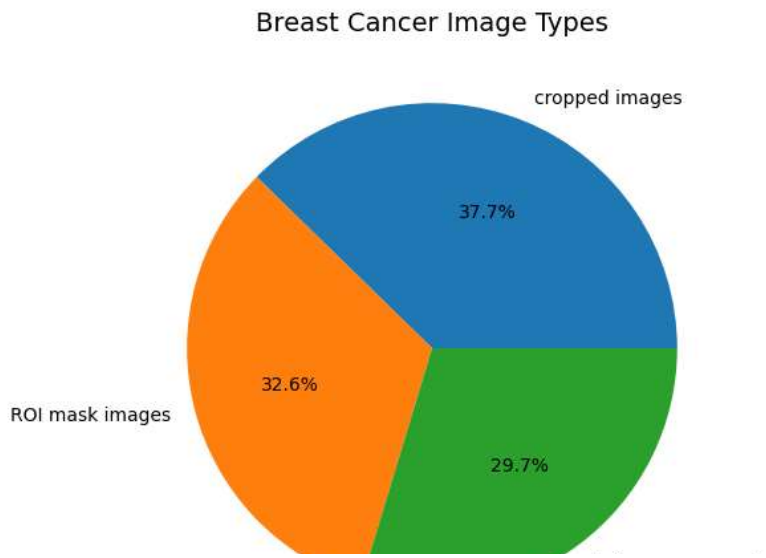


Image Enhancement

```
import tensorflow as tf
import cv2
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import plot_model


def image_processor(image_path, target_size):
    absolute_image_path = os.path.abspath(image_path)
    image = cv2.imread(absolute_image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (target_size[1], target_size[0]))
    image_array = image / 255.0
    return image_array


# Define the target size
target_size = (224, 224, 3)

# Apply preprocessor to train data
mass_data['processed images'] = mass_data['image file path'].apply(lambda x: image_processor(x, target_size))

# Create a binary mapper
class_mapper = {'MALIGNANT': 1, 'BENIGN': 0, 'BENIGN_WITHOUT_CALLBACK': 0}

# Convert the processed_images column to an array
X_resized = np.array(mass_data['processed images'].tolist())

# Apply class mapper to pathology column
mass_data['labels'] = mass_data['pathology'].replace(class_mapper)

# Check the number of classes
num_classes = len(mass_data['labels'].unique())
```

Preparing the Train/Test Set

```
# Split data into train, test, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(X_resized, mass_data['labels'].values, test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.33, random_state=42)

# Convert integer labels to one-hot encoded labels
y_train = to_categorical(y_train, num_classes)
```

```
y_test = to_categorical(y_test, num_classes)
y_val = to_categorical(y_val, num_classes)


print("X_train: ", X_train.shape)
print("y_train:", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test: ", y_test.shape)
print("X_val: ", X_val.shape)
print("y_val: ", y_val.shape)
```

```
X_train:  (1187, 224, 224, 3)
y_train: (1187, 2)
X_test:  (341, 224, 224, 3)
y_test:  (341, 2)
X_val:  (168, 224, 224, 3)
y_val:  (168, 2)
```

```
# Convert one-hot encoded labels to single values for CNN Model
y_train = np.argmax(y_train, axis=1)
y_val = np.argmax(y_val, axis=1)
y_test = np.argmax(y_test, axis=1)


print("y_train:", y_train.shape)
print("y_test: ", y_test.shape)
print("y_val: ", y_val.shape)
```

```
y_train: (1187,)
y_test:  (341,)
y_val:  (168,)
```

CNN Model

```
from keras.models import Sequential
from keras import layers, models
from keras.optimizers import RMSprop, Adam
from keras.layers import Flatten, Dense, Embedding


model0 = models.Sequential()

model0.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model0.add(layers.MaxPooling2D((2, 2)))
model0.add(layers.Conv2D(64, (3, 3), activation='relu'))
model0.add(layers.MaxPooling2D((2, 2)))
model0.add(layers.Conv2D(128, (3, 3), activation='relu'))
model0.add(layers.MaxPooling2D((2, 2)))

model0.add(layers.Flatten())
model0.add(layers.Dense(48, activation='relu'))
model0.add(layers.Dropout(0.5))
model0.add(layers.Dense(1, activation='sigmoid'))

model0.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 flatten (Flatten)           (None, 86528)             0
```

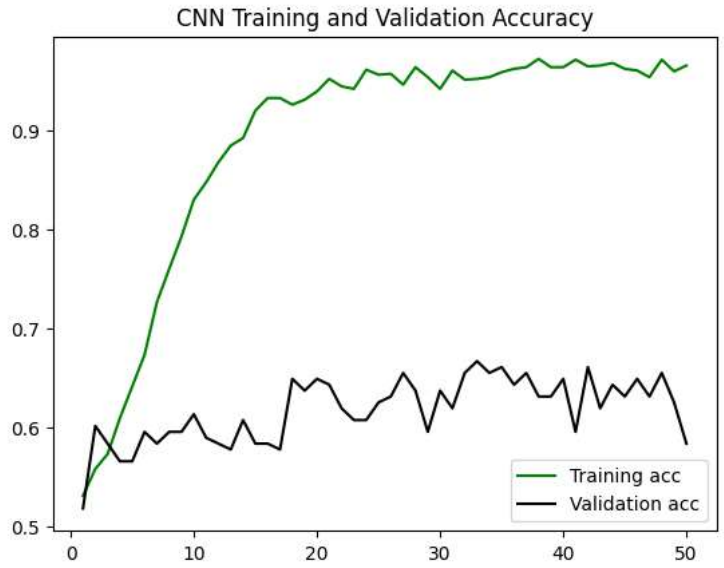
```
 dense (Dense)               (None, 48)                4153392

 dropout (Dropout)           (None, 48)                0

 dense_1 (Dense)             (None, 1)                 49

=================================================================
Total params: 4246689 (16.20 MB)
Trainable params: 4246689 (16.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

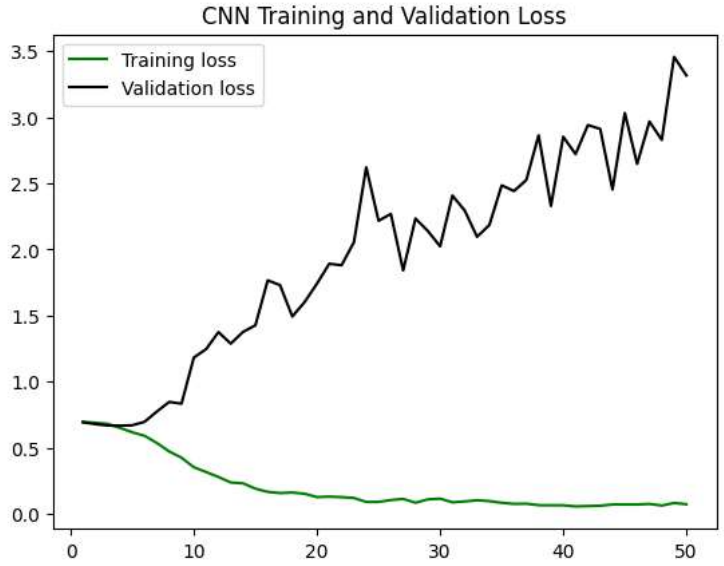
```
model0.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history0 = model0.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val))

test_loss, test_acc = model0.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 23/50
38/38 [==============================] - 2s 51ms/step - loss: 0.1186 - accuracy: 0.9419 - val_loss: 2.0542 - val_accuracy: 0.6071
Epoch 24/50
38/38 [==============================] - 2s 51ms/step - loss: 0.0883 - accuracy: 0.9612 - val_loss: 2.6212 - val_accuracy: 0.6071
Epoch 25/50
38/38 [==============================] - 2s 52ms/step - loss: 0.0891 - accuracy: 0.9562 - val_loss: 2.2166 - val_accuracy: 0.6250
Epoch 26/50
38/38 [==============================] - 2s 55ms/step - loss: 0.1027 - accuracy: 0.9570 - val_loss: 2.2684 - val_accuracy: 0.6310
Epoch 27/50
38/38 [==============================] - 2s 56ms/step - loss: 0.1117 - accuracy: 0.9461 - val_loss: 1.8412 - val_accuracy: 0.6548
Epoch 28/50
38/38 [==============================] - 2s 54ms/step - loss: 0.0829 - accuracy: 0.9638 - val_loss: 2.2334 - val_accuracy: 0.6369
Epoch 29/50
38/38 [==============================] - 2s 51ms/step - loss: 0.1079 - accuracy: 0.9537 - val_loss: 2.1398 - val_accuracy: 0.5952
Epoch 30/50
38/38 [==============================] - 2s 50ms/step - loss: 0.1133 - accuracy: 0.9419 - val_loss: 2.0230 - val_accuracy: 0.6369
Epoch 31/50
38/38 [==============================] - 2s 51ms/step - loss: 0.0860 - accuracy: 0.9604 - val_loss: 2.4073 - val_accuracy: 0.6190
Epoch 32/50
38/38 [==============================] - 2s 52ms/step - loss: 0.0923 - accuracy: 0.9511 - val_loss: 2.2938 - val_accuracy: 0.6548
Epoch 33/50
38/38 [==============================] - 2s 54ms/step - loss: 0.1015 - accuracy: 0.9520 - val_loss: 2.0950 - val_accuracy: 0.6667
Epoch 34/50
38/38 [==============================] - 2s 56ms/step - loss: 0.0950 - accuracy: 0.9537 - val_loss: 2.1845 - val_accuracy: 0.6548
Epoch 35/50
38/38 [==============================] - 2s 57ms/step - loss: 0.0822 - accuracy: 0.9587 - val_loss: 2.4846 - val_accuracy: 0.6607
Epoch 36/50
38/38 [==============================] - 2s 54ms/step - loss: 0.0740 - accuracy: 0.9621 - val_loss: 2.4416 - val_accuracy: 0.6429
Epoch 37/50
38/38 [==============================] - 2s 51ms/step - loss: 0.0753 - accuracy: 0.9638 - val_loss: 2.5250 - val_accuracy: 0.6548
Epoch 38/50
38/38 [==============================] - 2s 50ms/step - loss: 0.0639 - accuracy: 0.9722 - val_loss: 2.8634 - val_accuracy: 0.6310
Epoch 39/50
38/38 [==============================] - 2s 52ms/step - loss: 0.0635 - accuracy: 0.9638 - val_loss: 2.3286 - val_accuracy: 0.6310
Epoch 40/50
38/38 [==============================] - 2s 51ms/step - loss: 0.0635 - accuracy: 0.9638 - val_loss: 2.8524 - val_accuracy: 0.6488
Epoch 41/50
38/38 [==============================] - 2s 51ms/step - loss: 0.0548 - accuracy: 0.9714 - val_loss: 2.7208 - val_accuracy: 0.5952
Epoch 42/50
38/38 [==============================] - 2s 56ms/step - loss: 0.0575 - accuracy: 0.9646 - val_loss: 2.9406 - val_accuracy: 0.6607
Epoch 43/50
38/38 [==============================] - 2s 59ms/step - loss: 0.0599 - accuracy: 0.9655 - val_loss: 2.9115 - val_accuracy: 0.6190
Epoch 44/50
38/38 [==============================] - 2s 54ms/step - loss: 0.0690 - accuracy: 0.9680 - val_loss: 2.4528 - val_accuracy: 0.6429
Epoch 45/50
38/38 [==============================] - 2s 51ms/step - loss: 0.0696 - accuracy: 0.9621 - val_loss: 3.0315 - val_accuracy: 0.6310
Epoch 46/50
38/38 [==============================] - 2s 52ms/step - loss: 0.0693 - accuracy: 0.9604 - val_loss: 2.6472 - val_accuracy: 0.6488
Epoch 47/50
38/38 [==============================] - 2s 51ms/step - loss: 0.0735 - accuracy: 0.9537 - val_loss: 2.9675 - val_accuracy: 0.6310
Epoch 48/50
38/38 [==============================] - 2s 51ms/step - loss: 0.0613 - accuracy: 0.9714 - val_loss: 2.8287 - val_accuracy: 0.6548
Epoch 49/50
38/38 [==============================] - 2s 52ms/step - loss: 0.0806 - accuracy: 0.9596 - val_loss: 3.4571 - val_accuracy: 0.6250
Epoch 50/50
38/38 [==============================] - 2s 58ms/step - loss: 0.0711 - accuracy: 0.9655 - val_loss: 3.3164 - val_accuracy: 0.5833
11/11 [==============================] - 1s 65ms/step - loss: 3.3923 - accuracy: 0.6334
Test accuracy: 0.633431077003479
```

```
acc = history0.history['accuracy']
val_acc = history0.history['val_accuracy']
loss = history0.history['loss']
```

```
val_loss = history0.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'g', label='Training acc')
plt.plot(epochs, val_acc, 'k', label='Validation acc')
plt.title('CNN Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'g', label='Training loss')
plt.plot(epochs, val_loss, 'k', label='Validation loss')
plt.title('CNN Training and Validation Loss')
plt.legend()

plt.show()
```



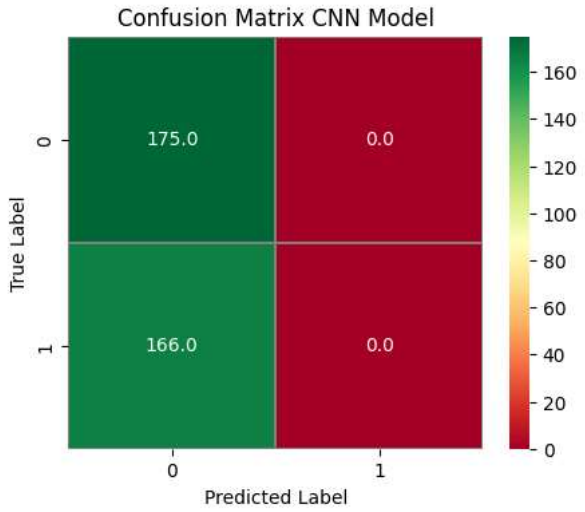



```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = model0.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = y_test

report0 = classification_report(y_true_classes, y_pred_classes)
print("Classification Report:")
print(report0)

cm0 = confusion_matrix(y_true_classes, y_pred_classes)
```

```
print("Confusion Matrix:")
print(cm0)
```

```
11/11 [==============================] - 0s 22ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.51      1.00      0.68       175
           1       0.00      0.00      0.00       166

    accuracy                           0.51       341
   macro avg       0.26      0.50      0.34       341
weighted avg       0.26      0.51      0.35       341

Confusion Matrix:
[[175   0]
 [166   0]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

```
f,ax = plt.subplots(figsize=(5, 4))
sns.heatmap(cm0, annot=True, linewidths=0.01,cmap="RdYlGn",linecolor="gray", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix CNN Model")
plt.show()
```



CNN Second Iteration

```
model1 = models.Sequential()

model1.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(64, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(128, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))

model1.add(layers.Flatten())
model1.add(layers.Dense(48, activation='relu'))
model1.add(layers.Dropout(0.5))
model1.add(layers.Dense(64, activation='relu'))   # Additional layer
model1.add(layers.Dense(128, activation='relu'))  # Additional layer
model1.add(layers.Dense(1, activation='sigmoid'))

model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 flatten (Flatten)           (None, 86528)             0

 dense (Dense)               (None, 48)                4153392

 dropout (Dropout)           (None, 48)                0

 dense_1 (Dense)             (None, 64)                3136

 dense_2 (Dense)             (None, 128)               8320

 dense_3 (Dense)             (None, 1)                 129

=================================================================
Total params: 4258225 (16.24 MB)
Trainable params: 4258225 (16.24 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history1 = model1.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_val, y_val))

test_loss, test_acc = model1.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 94/100
38/38 [==============================] - 2s 58ms/step - loss: 0.0234 - accuracy: 0.9857 - val_loss: 4.5368 - val_accuracy: 0.5893
Epoch 95/100
38/38 [==============================] - 2s 59ms/step - loss: 0.0436 - accuracy: 0.9798 - val_loss: 3.9849 - val_accuracy: 0.6071
Epoch 96/100
38/38 [==============================] - 2s 52ms/step - loss: 0.0424 - accuracy: 0.9832 - val_loss: 4.6946 - val_accuracy: 0.5952
Epoch 97/100
38/38 [==============================] - 2s 51ms/step - loss: 0.0373 - accuracy: 0.9832 - val_loss: 3.9983 - val_accuracy: 0.6369
Epoch 98/100
38/38 [==============================] - 2s 53ms/step - loss: 0.0330 - accuracy: 0.9874 - val_loss: 4.4225 - val_accuracy: 0.5952
Epoch 99/100
38/38 [==============================] - 2s 52ms/step - loss: 0.0325 - accuracy: 0.9806 - val_loss: 3.9140 - val_accuracy: 0.6012
Epoch 100/100
38/38 [==============================] - 2s 52ms/step - loss: 0.0225 - accuracy: 0.9840 - val_loss: 4.8677 - val_accuracy: 0.6131
11/11 [==============================] - 1s 64ms/step - loss: 4.6356 - accuracy: 0.6276
Test accuracy: 0.6275659799575806
```

Resnet50

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam


resnet50_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in resnet50_model.layers:
    layer.trainable = False

model2 = models.Sequential()
model2.add(resnet50_model)
model2.add(layers.GlobalAveragePooling2D())
model2.add(layers.Dense(256, activation='relu'))
model2.add(layers.Dense(128, activation='relu'))
model2.add(layers.Dense(64, activation='relu'))
model2.add(layers.Dropout(0.5))
model2.add(layers.Dense(2, activation='sigmoid'))

model2.summary()
```

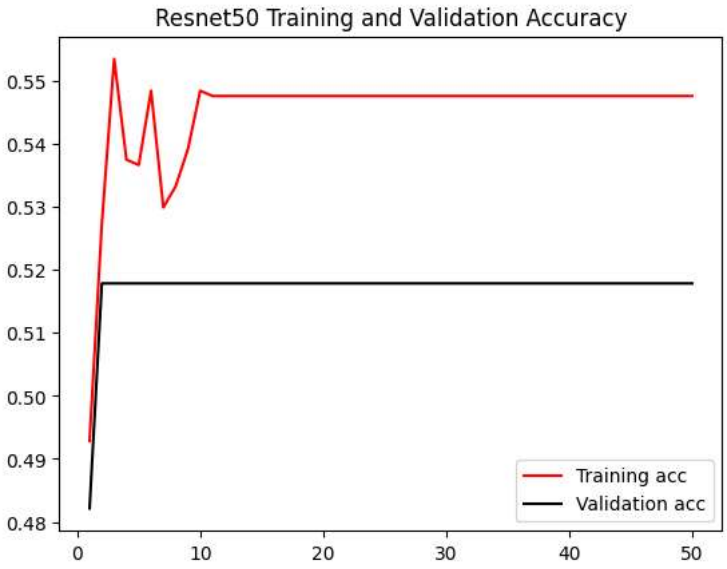
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 256)               524544

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dropout (Dropout)           (None, 64)                0

 dense_3 (Dense)             (None, 2)                 130

=================================================================
Total params: 24153538 (92.14 MB)
Trainable params: 565826 (2.16 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

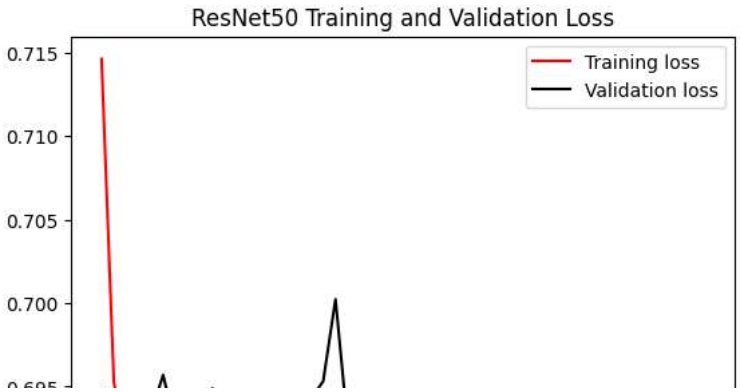
```
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history2 = model2.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val))

test_loss, test_acc = model2.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/50
38/38 [==============================] - 23s 200ms/step - loss: 0.7146 - accuracy: 0.4928 - val_loss: 0.6948 - val_accuracy: 0.4821
Epoch 2/50
38/38 [==============================] - 4s 112ms/step - loss: 0.6953 - accuracy: 0.5274 - val_loss: 0.6930 - val_accuracy: 0.5179
Epoch 3/50
38/38 [==============================] - 4s 108ms/step - loss: 0.6914 - accuracy: 0.5535 - val_loss: 0.6928 - val_accuracy: 0.5179
```

```
Epoch 4/50
38/38 [==============================] - 4s 108ms/step - loss: 0.6922 - accuracy: 0.5375 - val_loss: 0.6925 - val_accuracy: 0.5179
Epoch 5/50
38/38 [==============================] - 4s 114ms/step - loss: 0.6906 - accuracy: 0.5366 - val_loss: 0.6933 - val_accuracy: 0.5179
Epoch 6/50
38/38 [==============================] - 4s 114ms/step - loss: 0.6910 - accuracy: 0.5484 - val_loss: 0.6957 - val_accuracy: 0.5179
Epoch 7/50
38/38 [==============================] - 4s 109ms/step - loss: 0.6931 - accuracy: 0.5299 - val_loss: 0.6925 - val_accuracy: 0.5179
Epoch 8/50
38/38 [==============================] - 4s 113ms/step - loss: 0.6934 - accuracy: 0.5333 - val_loss: 0.6934 - val_accuracy: 0.5179
Epoch 9/50
38/38 [==============================] - 4s 116ms/step - loss: 0.6917 - accuracy: 0.5392 - val_loss: 0.6937 - val_accuracy: 0.5179
Epoch 10/50
38/38 [==============================] - 4s 113ms/step - loss: 0.6906 - accuracy: 0.5484 - val_loss: 0.6949 - val_accuracy: 0.5179
Epoch 11/50
38/38 [==============================] - 4s 109ms/step - loss: 0.6917 - accuracy: 0.5476 - val_loss: 0.6929 - val_accuracy: 0.5179
Epoch 12/50
38/38 [==============================] - 4s 116ms/step - loss: 0.6902 - accuracy: 0.5476 - val_loss: 0.6927 - val_accuracy: 0.5179
Epoch 13/50
38/38 [==============================] - 4s 118ms/step - loss: 0.6894 - accuracy: 0.5476 - val_loss: 0.6930 - val_accuracy: 0.5179
Epoch 14/50
38/38 [==============================] - 4s 110ms/step - loss: 0.6899 - accuracy: 0.5476 - val_loss: 0.6933 - val_accuracy: 0.5179
Epoch 15/50
38/38 [==============================] - 4s 110ms/step - loss: 0.6907 - accuracy: 0.5476 - val_loss: 0.6926 - val_accuracy: 0.5179
Epoch 16/50
38/38 [==============================] - 4s 115ms/step - loss: 0.6892 - accuracy: 0.5476 - val_loss: 0.6930 - val_accuracy: 0.5179
Epoch 17/50
38/38 [==============================] - 4s 116ms/step - loss: 0.6886 - accuracy: 0.5476 - val_loss: 0.6935 - val_accuracy: 0.5179
Epoch 18/50
38/38 [==============================] - 4s 111ms/step - loss: 0.6877 - accuracy: 0.5476 - val_loss: 0.6943 - val_accuracy: 0.5179
Epoch 19/50
38/38 [==============================] - 4s 114ms/step - loss: 0.6923 - accuracy: 0.5476 - val_loss: 0.6953 - val_accuracy: 0.5179
Epoch 20/50
38/38 [==============================] - 5s 120ms/step - loss: 0.6912 - accuracy: 0.5476 - val_loss: 0.7002 - val_accuracy: 0.5179
Epoch 21/50
38/38 [==============================] - 4s 111ms/step - loss: 0.6883 - accuracy: 0.5476 - val_loss: 0.6926 - val_accuracy: 0.5179
Epoch 22/50
38/38 [==============================] - 4s 114ms/step - loss: 0.6892 - accuracy: 0.5476 - val_loss: 0.6937 - val_accuracy: 0.5179
Epoch 23/50
38/38 [==============================] - 4s 117ms/step - loss: 0.6906 - accuracy: 0.5476 - val_loss: 0.6931 - val_accuracy: 0.5179
Epoch 24/50
38/38 [==============================] - 4s 115ms/step - loss: 0.6892 - accuracy: 0.5476 - val_loss: 0.6940 - val_accuracy: 0.5179
Epoch 25/50
38/38 [==============================] - 4s 114ms/step - loss: 0.6898 - accuracy: 0.5476 - val_loss: 0.6938 - val_accuracy: 0.5179
Epoch 26/50
38/38 [==============================] - 4s 116ms/step - loss: 0.6890 - accuracy: 0.5476 - val_loss: 0.6934 - val_accuracy: 0.5179
Epoch 27/50
38/38 [==============================] - 5s 121ms/step - loss: 0.6899 - accuracy: 0.5476 - val_loss: 0.6934 - val_accuracy: 0.5179
Epoch 28/50
38/38 [==============================] - 4s 113ms/step - loss: 0.6891 - accuracy: 0.5476 - val_loss: 0.6939 - val_accuracy: 0.5179
Epoch 29/50
38/38 [==============================] - 4s 112ms/step - loss: 0.6893 - accuracy: 0.5476 - val_loss: 0.6936 - val_accuracy: 0.5179
```

```
acc = history2.history['accuracy']
val_acc = history2.history['val_accuracy']
loss = history2.history['loss']
val_loss = history2.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'k', label='Validation acc')
plt.title('Resnet50 Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'k', label='Validation loss')
plt.title('ResNet50 Training and Validation Loss')
plt.legend()

plt.show()
```



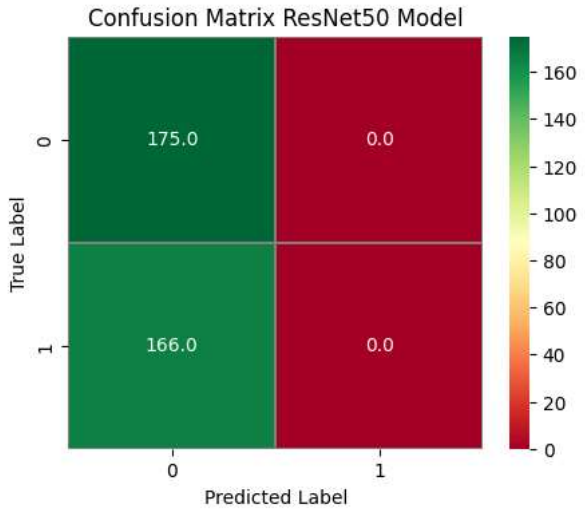



```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = model2.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

report2 = classification_report(y_true_classes, y_pred_classes)
print("Classification Report:")
print(report2)

cm2 = confusion_matrix(y_true_classes, y_pred_classes)
print("Confusion Matrix:")
print(cm2)
```

```
11/11 [==============================] - 3s 86ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.51      1.00      0.68       175
           1       0.00      0.00      0.00       166

    accuracy                           0.51       341
   macro avg       0.26      0.50      0.34       341
weighted avg       0.26      0.51      0.35       341

Confusion Matrix:
[[175   0]
 [166   0]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

```
f,ax = plt.subplots(figsize=(5, 4))
sns.heatmap(cm2, annot=True, linewidths=0.01,cmap="RdYlGn",linecolor="gray", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
```

```
plt.ylabel("True Label")
plt.title("Confusion Matrix ResNet50 Model")
plt.show()
```



ResNet50 Second Iteration

```
resnet50_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in resnet50_model.layers:
    layer.trainable = False

model3 = models.Sequential()
model3.add(resnet50_model)
model3.add(layers.GlobalAveragePooling2D())
model3.add(layers.Dense(256, activation='relu'))
model3.add(layers.Dropout(0.5))
model3.add(layers.Dense(128, activation='relu'))
model3.add(layers.Dense(64, activation='relu'))
model3.add(layers.Dense(32, activation='relu'))
model3.add(layers.Dense(16, activation='relu'))
model3.add(layers.Dense(1, activation='softmax'))

model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 256)               524544

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 2)                 34

=================================================================
Total params: 24156050 (92.15 MB)
Trainable params: 568338 (2.17 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

```
model3.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history3 = model3.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_val, y_val))

test_loss, test_acc = model3.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 47/100
38/38 [==============================] - 5s 125ms/step - loss: 0.6885 - accuracy: 0.5459 - val_loss: 0.6928 - val_accuracy: 0.5179
Epoch 48/100
38/38 [==============================] - 8s 223ms/step - loss: 0.6889 - accuracy: 0.5451 - val_loss: 0.6930 - val_accuracy: 0.5179
Epoch 49/100
38/38 [==============================] - 5s 126ms/step - loss: 0.6901 - accuracy: 0.5484 - val_loss: 0.6929 - val_accuracy: 0.5179
Epoch 50/100
38/38 [==============================] - 4s 119ms/step - loss: 0.6897 - accuracy: 0.5484 - val_loss: 0.6930 - val_accuracy: 0.5179
Epoch 51/100
38/38 [==============================] - 4s 116ms/step - loss: 0.6898 - accuracy: 0.5476 - val_loss: 0.6928 - val_accuracy: 0.5179
Epoch 52/100
38/38 [==============================] - 4s 117ms/step - loss: 0.6894 - accuracy: 0.5476 - val_loss: 0.6931 - val_accuracy: 0.5179
Epoch 53/100
38/38 [==============================] - 5s 122ms/step - loss: 0.6893 - accuracy: 0.5468 - val_loss: 0.6925 - val_accuracy: 0.5179
Epoch 54/100
38/38 [==============================] - 4s 119ms/step - loss: 0.6891 - accuracy: 0.5476 - val_loss: 0.6926 - val_accuracy: 0.5179
Epoch 55/100
38/38 [==============================] - 4s 118ms/step - loss: 0.6903 - accuracy: 0.5459 - val_loss: 0.6923 - val_accuracy: 0.5179
Epoch 56/100
38/38 [==============================] - 8s 224ms/step - loss: 0.6898 - accuracy: 0.5451 - val_loss: 0.6929 - val_accuracy: 0.5179
Epoch 57/100
38/38 [==============================] - 4s 114ms/step - loss: 0.6902 - accuracy: 0.5451 - val_loss: 0.6928 - val_accuracy: 0.5179
Epoch 58/100
38/38 [==============================] - 9s 229ms/step - loss: 0.6889 - accuracy: 0.5476 - val_loss: 0.6927 - val_accuracy: 0.5179
Epoch 59/100
38/38 [==============================] - 4s 117ms/step - loss: 0.6896 - accuracy: 0.5468 - val_loss: 0.6926 - val_accuracy: 0.5179
Epoch 60/100
38/38 [==============================] - 6s 162ms/step - loss: 0.6882 - accuracy: 0.5468 - val_loss: 0.6930 - val_accuracy: 0.5179
Epoch 61/100
38/38 [==============================] - 5s 121ms/step - loss: 0.6885 - accuracy: 0.5476 - val_loss: 0.6932 - val_accuracy: 0.5179
Epoch 62/100
38/38 [==============================] - 4s 114ms/step - loss: 0.6873 - accuracy: 0.5510 - val_loss: 0.6931 - val_accuracy: 0.5179
Epoch 63/100
38/38 [==============================] - 4s 115ms/step - loss: 0.6895 - accuracy: 0.5484 - val_loss: 0.6936 - val_accuracy: 0.5179
Epoch 64/100
38/38 [==============================] - 5s 122ms/step - loss: 0.6908 - accuracy: 0.5476 - val_loss: 0.6928 - val_accuracy: 0.5179
Epoch 65/100
38/38 [==============================] - 5s 121ms/step - loss: 0.6897 - accuracy: 0.5476 - val_loss: 0.6924 - val_accuracy: 0.5179
Epoch 66/100
38/38 [==============================] - 4s 116ms/step - loss: 0.6886 - accuracy: 0.5476 - val_loss: 0.6925 - val_accuracy: 0.5179
Epoch 67/100
38/38 [==============================] - 5s 119ms/step - loss: 0.6886 - accuracy: 0.5484 - val_loss: 0.6925 - val_accuracy: 0.5179
Epoch 68/100
38/38 [==============================] - 5s 122ms/step - loss: 0.6870 - accuracy: 0.5493 - val_loss: 0.6929 - val_accuracy: 0.5179
Epoch 69/100
38/38 [==============================] - 4s 119ms/step - loss: 0.6893 - accuracy: 0.5484 - val_loss: 0.6932 - val_accuracy: 0.5179
Epoch 70/100
38/38 [==============================] - 4s 116ms/step - loss: 0.6879 - accuracy: 0.5484 - val_loss: 0.6931 - val_accuracy: 0.5179
Epoch 71/100
38/38 [==============================] - 184s 5s/step - loss: 0.6893 - accuracy: 0.5501 - val_loss: 0.6931 - val_accuracy: 0.5179
Epoch 72/100
38/38 [==============================] - 87s 2s/step - loss: 0.6908 - accuracy: 0.5468 - val_loss: 0.6930 - val_accuracy: 0.5179
Epoch 73/100
38/38 [==============================] - 7s 184ms/step - loss: 0.6888 - accuracy: 0.5468 - val_loss: 0.6924 - val_accuracy: 0.5179
Epoch 74/100
38/38 [==============================] - 4s 116ms/step - loss: 0.6890 - accuracy: 0.5484 - val_loss: 0.6932 - val_accuracy: 0.5179
Epoch 75/100
38/38 [==============================] - 4s 119ms/step - loss: 0.6892 - accuracy: 0.5484 - val_loss: 0.6931 - val_accuracy: 0.5179
Epoch 76/100
```

CNN Model VGG16

```
from keras.applications import vgg16


vgg16_model = tf.keras.applications.vgg16.VGG16(
    weights = 'imagenet',
    include_top = False,
    input_shape = (224, 224, 3)
)

for layer in vgg16_model.layers:
    layer.trainable = False

model4 = tf.keras.Sequential()
```

```
model4.add(vgg16_model)
model4.add(tf.keras.layers.GlobalAveragePooling2D())
model4.add(tf.keras.layers.Dropout(0.5))
model4.add(tf.keras.layers.Dense(256, activation = 'relu'))
model4.add(tf.keras.layers.Dense(128, activation = 'relu'))
model4.add(tf.keras.layers.Dense(64, activation = 'relu'))
model4.add(tf.keras.layers.Dense(2, activation = 'softmax'))

model4.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d_1  (None, 512)               0
  (GlobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 256)               131328

 dense_5 (Dense)             (None, 128)               32896

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 2)                 130

=================================================================
Total params: 14887298 (56.79 MB)
Trainable params: 172610 (674.26 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
model4.compile(optimizer = 'adam', loss='binary_crossentropy',metrics=['accuracy'])

history4 = model4.fit(X_train, y_train, epochs = 50, batch_size = 32,  validation_data=(X_val, y_val))

test_loss, test_acc = model4.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

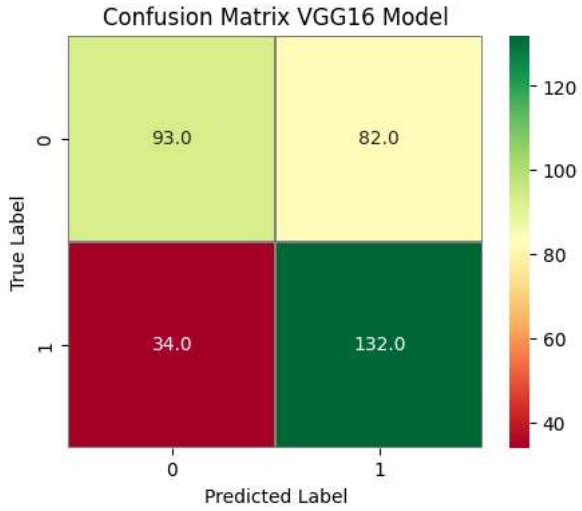
```
38/38 [==============================] - 6s 167ms/step - loss: 0.6446 - accuracy: 0.6352 - val_loss: 0.6272 - val_accuracy: 0.6190
Epoch 43/50
38/38 [==============================] - 6s 169ms/step - loss: 0.6418 - accuracy: 0.6276 - val_loss: 0.6185 - val_accuracy: 0.6369
Epoch 44/50
38/38 [==============================] - 6s 151ms/step - loss: 0.6459 - accuracy: 0.6259 - val_loss: 0.6288 - val_accuracy: 0.6012
Epoch 45/50
38/38 [==============================] - 6s 152ms/step - loss: 0.6448 - accuracy: 0.6369 - val_loss: 0.6307 - val_accuracy: 0.6548
Epoch 46/50
38/38 [==============================] - 7s 177ms/step - loss: 0.6429 - accuracy: 0.6327 - val_loss: 0.6238 - val_accuracy: 0.6488
Epoch 47/50
38/38 [==============================] - 7s 174ms/step - loss: 0.6369 - accuracy: 0.6369 - val_loss: 0.6271 - val_accuracy: 0.6250
Epoch 48/50
38/38 [==============================] - 7s 181ms/step - loss: 0.6440 - accuracy: 0.6293 - val_loss: 0.6252 - val_accuracy: 0.6488
Epoch 49/50
38/38 [==============================] - 8s 199ms/step - loss: 0.6449 - accuracy: 0.6209 - val_loss: 0.6542 - val_accuracy: 0.6429
Epoch 50/50
38/38 [==============================] - 6s 151ms/step - loss: 0.6432 - accuracy: 0.6175 - val_loss: 0.6254 - val_accuracy: 0.6548
11/11 [==============================] - 4s 379ms/step - loss: 0.6306 - accuracy: 0.6598
Test accuracy: 0.6598240733146667
```

```
acc = history4.history['accuracy']
val_acc = history4.history['val_accuracy']
loss = history4.history['loss']
val_loss = history4.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'g', label='Validation acc')
plt.title('VGG16 Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('VGG16 Training and Validation Loss')
plt.legend()

plt.show()
```


```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = model4.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

report4 = classification_report(y_true_classes, y_pred_classes)
print("Classification Report:")
print(report4)

cm4 = confusion_matrix(y_true_classes, y_pred_classes)
print("Confusion Matrix:")
print(cm4)
```

```
11/11 [==============================] - 2s 133ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.53      0.62       175
           1       0.62      0.80      0.69       166

    accuracy                           0.66       341
   macro avg       0.67      0.66      0.66       341
weighted avg       0.68      0.66      0.65       341

Confusion Matrix:
[[ 93  82]
 [ 34 132]]
```

```
f,ax = plt.subplots(figsize=(5, 4))
sns.heatmap(cm4, annot=True, linewidths=0.01,cmap="RdYlGn",linecolor="gray", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix VGG16 Model")
plt.show()
```



Second Iteration of VGG16 Model

```
vgg16_model = tf.keras.applications.vgg16.VGG16(
    weights = 'imagenet',
    include_top = False,
    input_shape = (224, 224, 3)
)

for layer in vgg16_model.layers:
```

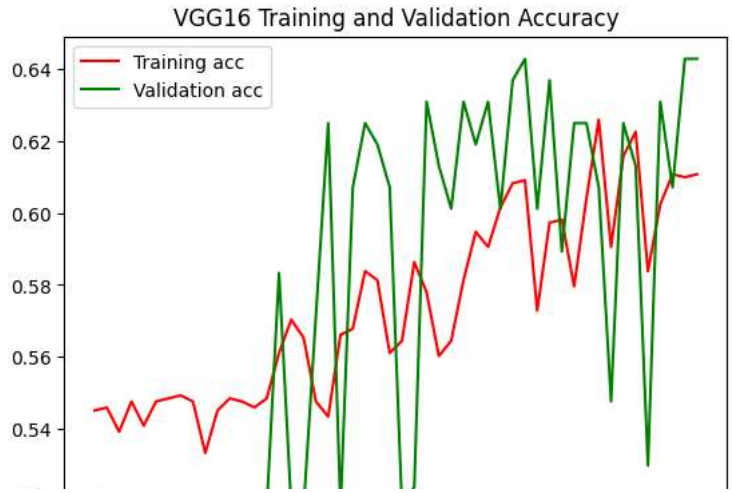
```
    layer.trainable = False

model5 = tf.keras.Sequential()
model5.add(vgg16_model)
model5.add(tf.keras.layers.GlobalAveragePooling2D())
model5.add(tf.keras.layers.Dropout(0.5))
model5.add(tf.keras.layers.Dense(256, activation = 'relu'))
model5.add(tf.keras.layers.Dense(128, activation='relu'))
model5.add(tf.keras.layers.Dense(64, activation='relu'))
model5.add(tf.keras.layers.Dense(32, activation='relu'))
model5.add(tf.keras.layers.Dense(16, activation='relu'))
model5.add(tf.keras.layers.Dense(2, activation = 'sigmoid'))

model5.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 256)               131328

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 2)                 34

=================================================================
Total params: 14889810 (56.80 MB)
Trainable params: 175122 (684.07 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
model5.compile(optimizer = 'rmsprop', loss='binary_crossentropy',metrics=['accuracy'])

history5 = model5.fit(X_train, y_train, epochs = 50, batch_size = 32,  validation_data=(X_val, y_val))

test_loss, test_acc = model5.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
38/38 [==============================] - 6s 172ms/step - loss: 0.6684 - accuracy: 0.6091 - val_loss: 0.6552 - val_accuracy: 0.6429
Epoch 37/50
38/38 [==============================] - 6s 167ms/step - loss: 0.6711 - accuracy: 0.5729 - val_loss: 0.6530 - val_accuracy: 0.6012
Epoch 38/50
38/38 [==============================] - 6s 168ms/step - loss: 0.6688 - accuracy: 0.5973 - val_loss: 0.6483 - val_accuracy: 0.6369
Epoch 39/50
38/38 [==============================] - 6s 170ms/step - loss: 0.6652 - accuracy: 0.5981 - val_loss: 0.6581 - val_accuracy: 0.5893
Epoch 40/50
38/38 [==============================] - 6s 150ms/step - loss: 0.6738 - accuracy: 0.5796 - val_loss: 0.6695 - val_accuracy: 0.6250
Epoch 41/50
38/38 [==============================] - 6s 155ms/step - loss: 0.6645 - accuracy: 0.6040 - val_loss: 0.6551 - val_accuracy: 0.6250
Epoch 42/50
38/38 [==============================] - 6s 153ms/step - loss: 0.6570 - accuracy: 0.6259 - val_loss: 0.6432 - val_accuracy: 0.6071
Epoch 43/50
38/38 [==============================] - 6s 168ms/step - loss: 0.6700 - accuracy: 0.5906 - val_loss: 0.6596 - val_accuracy: 0.5476
Epoch 44/50
38/38 [==============================] - 6s 156ms/step - loss: 0.6578 - accuracy: 0.6158 - val_loss: 0.6539 - val_accuracy: 0.6250
Epoch 45/50
38/38 [==============================] - 6s 169ms/step - loss: 0.6634 - accuracy: 0.6226 - val_loss: 0.6463 - val_accuracy: 0.6131
Epoch 46/50
38/38 [==============================] - 6s 167ms/step - loss: 0.6671 - accuracy: 0.5838 - val_loss: 0.6642 - val_accuracy: 0.5298
Epoch 47/50
38/38 [==============================] - 6s 171ms/step - loss: 0.6705 - accuracy: 0.6024 - val_loss: 0.6595 - val_accuracy: 0.6310
Epoch 48/50
38/38 [==============================] - 6s 167ms/step - loss: 0.6635 - accuracy: 0.6108 - val_loss: 0.6477 - val_accuracy: 0.6071
Epoch 49/50
38/38 [==============================] - 6s 168ms/step - loss: 0.6599 - accuracy: 0.6099 - val_loss: 0.6571 - val_accuracy: 0.6429
Epoch 50/50
38/38 [==============================] - 6s 156ms/step - loss: 0.6606 - accuracy: 0.6108 - val_loss: 0.6370 - val_accuracy: 0.6429
11/11 [==============================] - 4s 388ms/step - loss: 0.6385 - accuracy: 0.6657
Test accuracy: 0.6656891703605652
```

```
acc = history5.history['accuracy']
val_acc = history5.history['val_accuracy']
loss = history5.history['loss']
val_loss = history5.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'g', label='Validation acc')
plt.title('VGG16 Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('VGG16 Training and Validation Loss')
plt.legend()

plt.show()
```




```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = model5.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

report5 = classification_report(y_true_classes, y_pred_classes)
print("Classification Report:")
print(report5)

cm5 = confusion_matrix(y_true_classes, y_pred_classes)
print("Confusion Matrix:")
print(cm5)
```

```
11/11 [==============================] - 2s 116ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.69      0.68       175
           1       0.66      0.64      0.65       166

    accuracy                           0.67       341
   macro avg       0.67      0.66      0.67       341
weighted avg       0.67      0.67      0.67       341

Confusion Matrix:
[[121  54]
 [ 60 106]]
```

```
f,ax = plt.subplots(figsize=(5, 4))
sns.heatmap(cm5, annot=True, linewidths=0.01,cmap="RdYlGn",linecolor="gray", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix VGG16 Model")
plt.show()
```