

Import Packages

```
import os
import numpy as np
import torch
from torchvision import datasets, transforms as T, models
from torch import nn, optim
from torch.utils.data import DataLoader
import warnings
warnings.filterwarnings("ignore")
from tqdm import tqdm
from sklearn.metrics import confusion_matrix,
accuracy_score, precision_score, recall_score, f1_score,
classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import auc
```

Download Dataset

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("adityamahimkar/iqothnccd-lung-
cancer-dataset/versions/1")

print("Path to dataset files:", path)

Downloading from
https://www.kaggle.com/api/v1/datasets/download/adityamahimkar/iqothnc
cd-lung-cancer-dataset?dataset_version_number=1...

100%|██████████| 149M/149M [00:47<00:00, 3.31MB/s]

Extracting model files...

Path to dataset files:
/Users/odai/.cache/kagglehub/datasets/adityamahimkar/iqothnccd-lung-
cancer-dataset/versions/1
```

Copy Dataset

```
import shutil
```

```

target_path = "./"
shutil.copytree(path, target_path, dirs_exist_ok=True)

'./'

from sklearn.model_selection import train_test_split
original_dataset_dir = 'The IQ-OTHNCCD lung cancer dataset'
base_dir = 'dataset'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
for dir in [train_dir, validation_dir, test_dir]:
    os.makedirs(dir, exist_ok=True)
train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2

```

Split the dataset

```

def split_and_copy_files(class_dir, dest_base_dir):
    all_images = [os.path.join(class_dir, f) for f in
os.listdir(class_dir) if os.path.isfile(os.path.join(class_dir, f))]
    train_images, temp_images =
train_test_split(all_images, test_size=(1 - train_ratio),
random_state=42)
    validation_images, test_images =
train_test_split(temp_images, test_size=test_ratio/(validation_ratio +
test_ratio), random_state=42)
    for images, folder_name in zip([train_images,
validation_images, test_images], ['train', 'validation', 'test']):
        dest_dir = os.path.join(dest_base_dir,
folder_name, os.path.basename(class_dir))
        os.makedirs(dest_dir, exist_ok=True)
        for file in images:
            shutil.copy(file, dest_dir)
# Apply to all class subfolders
xx=os.listdir(original_dataset_dir)

for class_dir in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_dir)
    print("Full:", full_class_dir)
    print("Original:", original_dataset_dir)
    if os.path.isdir(full_class_dir):
        split_and_copy_files(full_class_dir, base_dir)

Full: The IQ-OTHNCCD lung cancer dataset/Bengin cases
Original: The IQ-OTHNCCD lung cancer dataset
Full: The IQ-OTHNCCD lung cancer dataset/.DS_Store
Original: The IQ-OTHNCCD lung cancer dataset
Full: The IQ-OTHNCCD lung cancer dataset/Normal cases
Original: The IQ-OTHNCCD lung cancer dataset
Full: The IQ-OTHNCCD lung cancer dataset/Malignant cases

```

Original: The IQ-OTHNCCD lung cancer dataset
Full: The IQ-OTHNCCD lung cancer dataset/IQ-OTH_NCCD lung cancer dataset.txt
Original: The IQ-OTHNCCD lung cancer dataset

Data Augmentation

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Flatten, Dense, Dropout
from tensorflow.keras.layers import Dense, Flatten,
Dropout, GlobalAveragePooling2D
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix,
accuracy_score, precision_score, recall_score, f1_score,
classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
def create_data_generators(train_dir, val_dir, test_dir, img_size=(240,
240), batch_size=32):
    # Data augmentation for training dataset
    train_datagen =
ImageDataGenerator(rescale=1./255, rotation_range=360, horizontal_flip=True,
vertical_flip=True, width_shift_range=0.1, height_shift_range=0.1)
    val_test_datagen = ImageDataGenerator(rescale=1./255)
    # Generate augmented training images
    train_generator =
train_datagen.flow_from_directory(train_dir, target_size=img_size, batch
_size=batch_size, class_mode='categorical', shuffle=True)
    # Generate validation images
    val_generator =
val_test_datagen.flow_from_directory(val_dir, target_size=img_size, batc
h_size=batch_size, class_mode='categorical', shuffle=False)
    # Generate test images
    test_generator =
val_test_datagen.flow_from_directory(test_dir, target_size=img_size, batc
h_size=batch_size, class_mode='categorical', shuffle=False)
    return train_generator, val_generator, test_generator
```

CNN Model

```
def build_cnn_model(input_shape, num_classes):
    model = Sequential([Conv2D(32, (3, 3),
activation='relu', input_shape=input_shape), MaxPooling2D(2, 2),
Conv2D(64, (3, 3),
activation='relu'), MaxPooling2D(2, 2), Conv2D(128, (3, 3),
```

```

activation='relu'),
                                MaxPooling2D(2, 2),Flatten(),Dense(128,
activation='relu'),Dropout(0.5),Dense(num_classes,
activation='softmax')
                                ])

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics
=['accuracy'])
    return model

```

```

# Directories for datasets
train_dir = 'dataset/train'
val_dir = 'dataset/validation'
test_dir = 'dataset/test'
# Create data generators
train_gen, val_gen, test_gen =
create_data_generators(train_dir, val_dir, test_dir)
# Get input shape and number of classes
input_shape = (240, 240, 3)
num_classes = len(train_gen.class_indices)
# Build CNN model
model = build_cnn_model(input_shape, num_classes)
# Train the model
history = model.fit(train_gen, epochs=10, validation_data=val_gen)

```

Found 766 images belonging to 3 classes.
Found 109 images belonging to 3 classes.
Found 222 images belonging to 3 classes.

2026-03-07 15:50:46.764565: I
tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow
binary is optimized with oneAPI Deep Neural Network Library (oneDNN)
to use the following CPU instructions in performance-critical
operations: SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.

```

Epoch 1/10
24/24 [=====] - 23s 906ms/step - loss: 1.4359
- accuracy: 0.4765 - val_loss: 0.9466 - val_accuracy: 0.5138
Epoch 2/10
24/24 [=====] - 24s 1s/step - loss: 0.9837 -
accuracy: 0.5026 - val_loss: 1.0047 - val_accuracy: 0.4220
Epoch 3/10
24/24 [=====] - 23s 946ms/step - loss: 0.9438
- accuracy: 0.5509 - val_loss: 0.8355 - val_accuracy: 0.5872
Epoch 4/10
24/24 [=====] - 25s 1s/step - loss: 0.9276 -
accuracy: 0.5209 - val_loss: 0.8555 - val_accuracy: 0.5229
Epoch 5/10

```

```

24/24 [=====] - 24s 973ms/step - loss: 0.9384
- accuracy: 0.5535 - val_loss: 0.8723 - val_accuracy: 0.5872
Epoch 6/10
24/24 [=====] - 27s 1s/step - loss: 0.9273 -
accuracy: 0.5104 - val_loss: 0.8557 - val_accuracy: 0.5872
Epoch 7/10
24/24 [=====] - 30s 1s/step - loss: 0.9217 -
accuracy: 0.5392 - val_loss: 0.7743 - val_accuracy: 0.7615
Epoch 8/10
24/24 [=====] - 27s 1s/step - loss: 0.8891 -
accuracy: 0.5966 - val_loss: 0.7749 - val_accuracy: 0.7615
Epoch 9/10
24/24 [=====] - 24s 992ms/step - loss: 0.8987
- accuracy: 0.5914 - val_loss: 0.7591 - val_accuracy: 0.5872
Epoch 10/10
24/24 [=====] - 23s 950ms/step - loss: 0.8953
- accuracy: 0.5888 - val_loss: 0.7197 - val_accuracy: 0.7706

```

```

def evaluate_model(model_name, y_true, y_pred):
    labels = list(test_gen.class_indices.keys())
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    print("Accuracy:", accuracy)
    print("Precision :", precision)
    print("Recall:", recall)
    print("F1-score:" , f1)
    print("Classification Report:")
    print(classification_report(y_true, y_pred_classes,
                                target_names=labels))
    # Plot confusion matrix
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.show()

```

```

y_pred=model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('CNN', y_true, y_pred_classes)

```

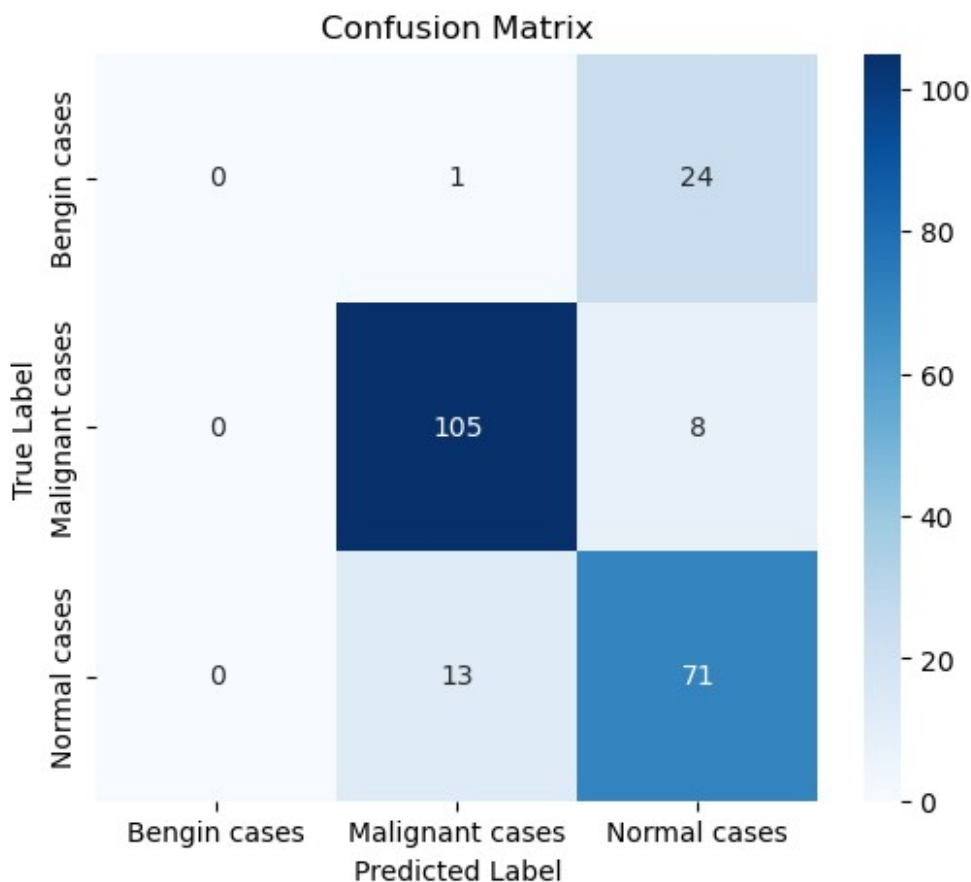
```

7/7 [=====] - 2s 259ms/step
Accuracy: 0.7927927927927928
Precision : 0.709949526911263

```

Recall: 0.7927927927927928
F1-score: 0.7480656712401135
Classification Report:

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.88	0.93	0.91	113
Normal cases	0.69	0.85	0.76	84
accuracy			0.79	222
macro avg	0.52	0.59	0.55	222
weighted avg	0.71	0.79	0.75	222



Resnet8

```
def build_resnet8_model(input_shape, num_classes):  
    base_model = ResNet50(weights='imagenet', include_top=False,  
                           input_shape=input_shape)  
    base_model.trainable = False # Freeze the base model  
    model = Sequential([  
        base_model,
```

```

GlobalAveragePooling2D(),
Dense(128, activation='relu'),
Dropout(0.5),
Dense(num_classes, activation='softmax')
])
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
return model

```

```

model = build_resnet8_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)

```

Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/resnet/>

resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5

94765736/94765736 [=====] - 4s 0us/step

Epoch 1/5

24/24 [=====] - 43s 2s/step - loss: 1.0463 - accuracy: 0.4621 - val_loss: 0.9433 - val_accuracy: 0.5138

Epoch 2/5

24/24 [=====] - 40s 2s/step - loss: 0.9801 - accuracy: 0.4883 - val_loss: 0.9293 - val_accuracy: 0.5138

Epoch 3/5

24/24 [=====] - 44s 2s/step - loss: 0.9883 - accuracy: 0.4530 - val_loss: 0.9311 - val_accuracy: 0.5138

Epoch 4/5

24/24 [=====] - 43s 2s/step - loss: 0.9694 - accuracy: 0.4909 - val_loss: 0.9332 - val_accuracy: 0.5138

Epoch 5/5

24/24 [=====] - 43s 2s/step - loss: 0.9583 - accuracy: 0.5170 - val_loss: 0.9260 - val_accuracy: 0.5138

```
y_pred=model.predict(test_gen)
```

```
y_true = test_gen.classes
```

```
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
evaluate_model('ResNet50', y_true, y_pred_classes)
```

7/7 [=====] - 11s 1s/step

Accuracy: 0.509009009009009

Precision : 0.2590901712523334

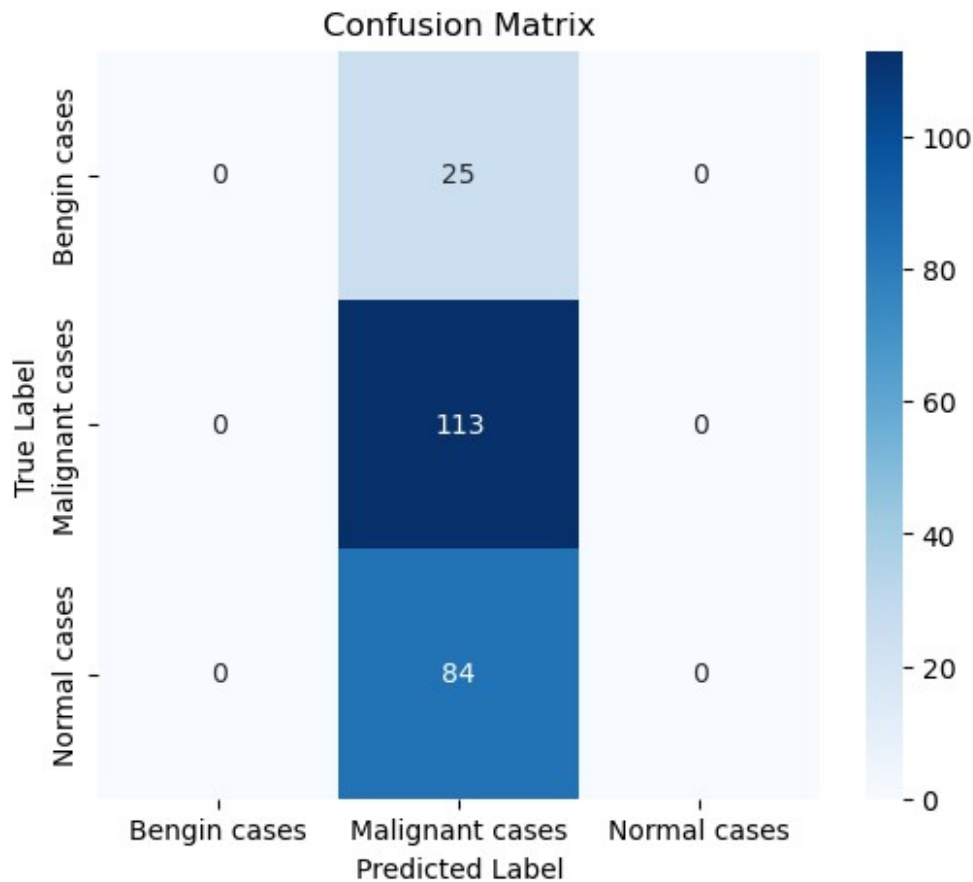
Recall: 0.509009009009009

F1-score: 0.34339115234637624

Classification Report:

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.51	1.00	0.67	113
Normal cases	0.00	0.00	0.00	84

accuracy				0.51	222
macro avg	0.17	0.33	0.22	0.22	222
weighted avg	0.26	0.51	0.34	0.34	222



Efficientnetb0

```

from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D,
Dense,Dropout
def build_efficientnetb0_model(input_shape, num_classes):
    base_model = EfficientNetB0(weights='imagenet',
include_top=False,input_shape=input_shape)
    base_model.trainable = False # Freeze the base model
    model = Sequential([base_model,GlobalAveragePooling2D(),Dense(128,
activation='relu'),Dropout(0.5),
                        Dense(num_classes, activation='softmax')
                        ])

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics
=['accuracy'])

```

```

return model
model = build_efficientnetb0_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)

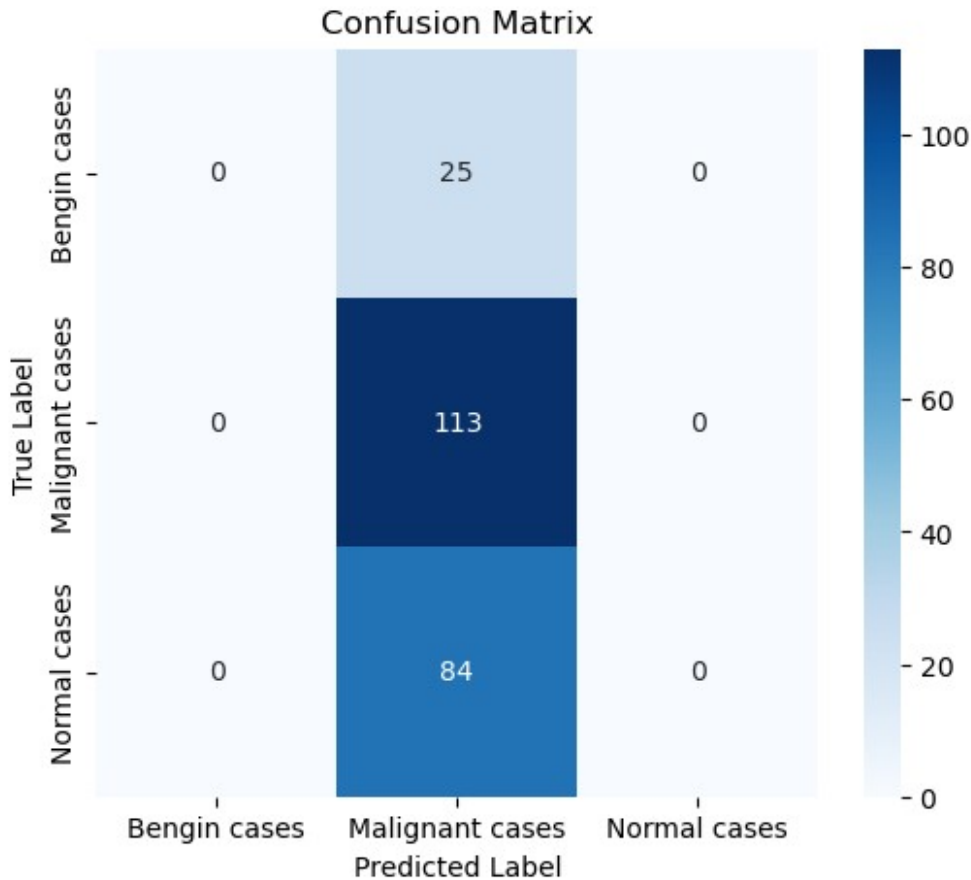
Downloading data from https://storage.googleapis.com/keras-
applications/efficientnetb0_notop.h5
16705208/16705208 [=====] - 2s 0us/step
Epoch 1/5
24/24 [=====] - 35s 1s/step - loss: 0.9998 -
accuracy: 0.4713 - val_loss: 0.9582 - val_accuracy: 0.5138
Epoch 2/5
24/24 [=====] - 28s 1s/step - loss: 0.9676 -
accuracy: 0.4974 - val_loss: 0.9541 - val_accuracy: 0.5138
Epoch 3/5
24/24 [=====] - 27s 1s/step - loss: 0.9717 -
accuracy: 0.5052 - val_loss: 0.9580 - val_accuracy: 0.5138
Epoch 4/5
24/24 [=====] - 27s 1s/step - loss: 0.9785 -
accuracy: 0.5078 - val_loss: 0.9545 - val_accuracy: 0.5138
Epoch 5/5
24/24 [=====] - 28s 1s/step - loss: 0.9720 -
accuracy: 0.4987 - val_loss: 0.9534 - val_accuracy: 0.5138

y_pred=model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('EfficientNetB0', y_true, y_pred_classes)

7/7 [=====] - 7s 857ms/step
Accuracy: 0.509009009009009
Precision : 0.2590901712523334
Recall: 0.509009009009009
F1-score: 0.34339115234637624
Classification Report:

```

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.51	1.00	0.67	113
Normal cases	0.00	0.00	0.00	84
accuracy			0.51	222
macro avg	0.17	0.33	0.22	222
weighted avg	0.26	0.51	0.34	222



Inceptionv3

```
from tensorflow.keras.applications import InceptionV3
def build_inceptionv3_model(input_shape, num_classes):
    base_model = InceptionV3(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False # Freeze the base model

    model = Sequential([base_model, GlobalAveragePooling2D(), Dense(128,
activation='relu'),
                        Dropout(0.5),
                        Dense(num_classes, activation='softmax')
                        ])

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])
    return model
model = build_inceptionv3_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)

Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/inception_v3/
inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
```

```

87910968/87910968 [=====] - 4s 0us/step
Epoch 1/5
24/24 [=====] - 29s 1s/step - loss: 0.8958 -
accuracy: 0.6749 - val_loss: 0.4255 - val_accuracy: 0.8532
Epoch 2/5
24/24 [=====] - 22s 924ms/step - loss: 0.5108
- accuracy: 0.8107 - val_loss: 0.3492 - val_accuracy: 0.8440
Epoch 3/5
24/24 [=====] - 23s 933ms/step - loss: 0.4317
- accuracy: 0.8446 - val_loss: 0.3324 - val_accuracy: 0.8532
Epoch 4/5
24/24 [=====] - 23s 948ms/step - loss: 0.4165
- accuracy: 0.8394 - val_loss: 0.3055 - val_accuracy: 0.8624
Epoch 5/5
24/24 [=====] - 23s 932ms/step - loss: 0.4233
- accuracy: 0.8355 - val_loss: 0.3088 - val_accuracy: 0.8716

```

```

y_pred=model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('InceptionV3(', y_true, y_pred_classes)

```

```

7/7 [=====] - 5s 602ms/step

```

```

Accuracy: 0.8693693693693694
Precision : 0.7873549224900577
Recall: 0.8693693693693694
F1-score: 0.8220262845262846
Classification Report:

```

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.99	0.97	0.98	113
Normal cases	0.75	0.99	0.85	84
accuracy			0.87	222
macro avg	0.58	0.65	0.61	222
weighted avg	0.79	0.87	0.82	222

