

all_lectures_clean

July 10, 2026

1 Lecture 1 — Basic Concepts of Image

Prepared by Dr. Hager Saleh | Midocean University

```
[1]: import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

DATASET_DIR = 'extracted_data/The IQ-OTHNCCD lung cancer dataset'
CLASSES = ['Bengin cases', 'Malignant cases', 'Normal cases']

# Pick one sample image from each class
sample_images = {}
for cls in CLASSES:
    folder = os.path.join(DATASET_DIR, cls)
    files = os.listdir(folder)
    sample_images[cls] = os.path.join(folder, files[0])

print('Sample images:', sample_images)
```

```
Sample images: {'Bengin cases': 'extracted_data/The IQ-OTHNCCD lung cancer
dataset/Bengin cases/Bengin case (85).jpg', 'Malignant cases':
'extracted_data/The IQ-OTHNCCD lung cancer dataset/Malignant cases/Malignant
case (257).jpg', 'Normal cases': 'extracted_data/The IQ-OTHNCCD lung cancer
dataset/Normal cases/Normal case (19).jpg'}
```

1.1 Read and Display Image as RGB

```
[2]: # Read and display one sample image as RGB
img_path = sample_images['Malignant cases']
image = cv2.imread(img_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image_rgb)
plt.title('Loaded Image (RGB)')
plt.axis('off')
plt.show()
```

```
# Get image dimensions
height, width, channels = image.shape
print(f'Width: {width}, Height: {height}, Channels: {channels}')
```

Loaded Image (RGB)

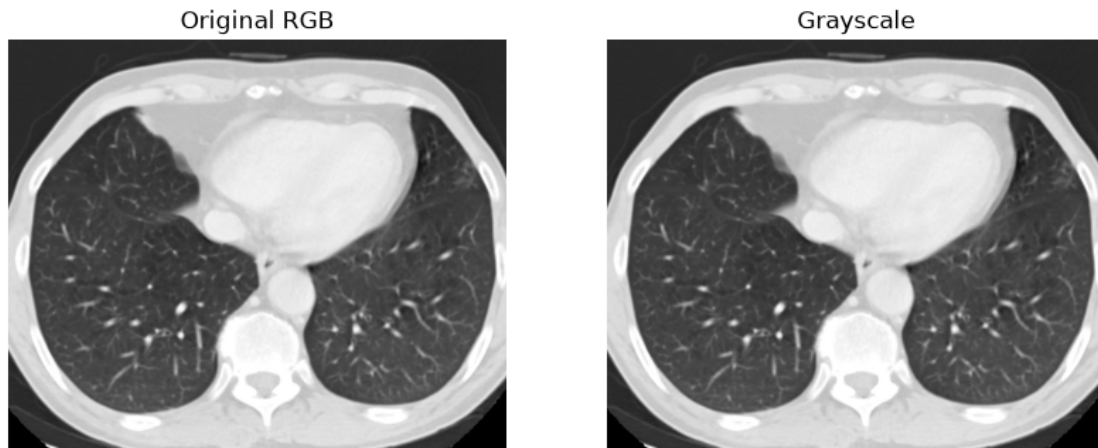


Width: 623, Height: 512, Channels: 3

1.2 Convert Image to Grayscale

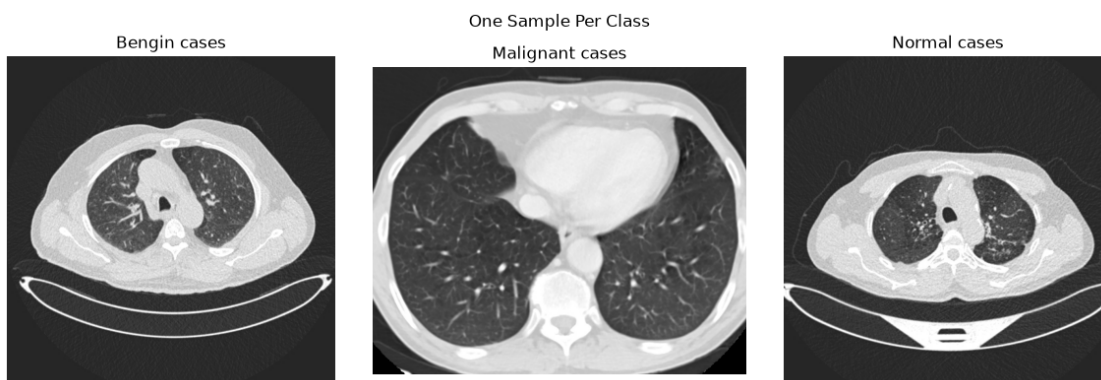
```
[3]: gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

fig, axes = plt.subplots(1, 2, figsize=(10, 4))
axes[0].imshow(image_rgb)
axes[0].set_title('Original RGB')
axes[0].axis('off')
axes[1].imshow(gray_image, cmap='gray')
axes[1].set_title('Grayscale')
axes[1].axis('off')
plt.show()
```



1.3 Display One Sample Per Class

```
[4]: fig, axes = plt.subplots(1, 3, figsize=(12, 4))
    for ax, (cls, path) in zip(axes, sample_images.items()):
        img = cv2.cvtColor(cv2.imread(path), cv2.COLOR_BGR2RGB)
        ax.imshow(img)
        ax.set_title(cls)
        ax.axis('off')
    plt.suptitle('One Sample Per Class')
    plt.tight_layout()
    plt.show()
```



1.4 Image Data Augmentation (Keras)

```
[5]: from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, \
      ↪img_to_array

def display_augmented_images(img_path, datagen, num_augmented=3, title=''):
    img = load_img(img_path)
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    augmented = datagen.flow(img_array, batch_size=1)

    plt.figure(figsize=(4 * (num_augmented + 1), 4))
    plt.subplot(1, num_augmented + 1, 1)
    plt.title('Original')
    plt.imshow(img)
    plt.axis('off')
    for i in range(num_augmented):
        batch = next(augmented)
        aug_img = batch[0].astype('uint8')
        plt.subplot(1, num_augmented + 1, i + 2)
        plt.title(f'Aug {i+1}')
        plt.imshow(aug_img)
        plt.axis('off')
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

sample_path = sample_images['Malignant cases']

# Horizontal Flip
display_augmented_images(sample_path, ImageDataGenerator(horizontal_flip=True), \
    ↪title='Horizontal Flip')

# Vertical Flip
display_augmented_images(sample_path, ImageDataGenerator(vertical_flip=True), \
    ↪title='Vertical Flip')

# Rotation
display_augmented_images(sample_path, ImageDataGenerator(rotation_range=40), \
    ↪title='Random Rotation')

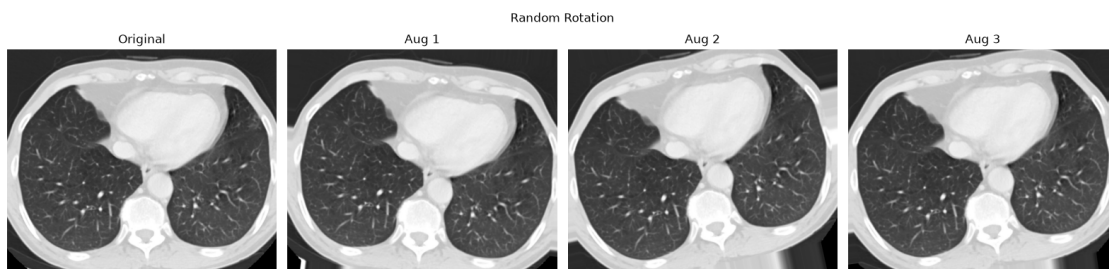
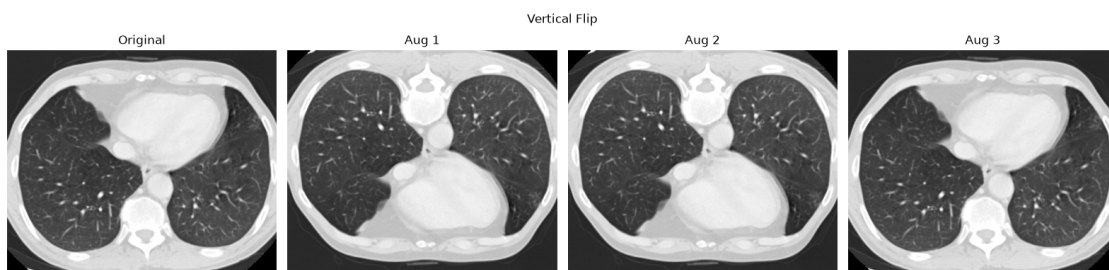
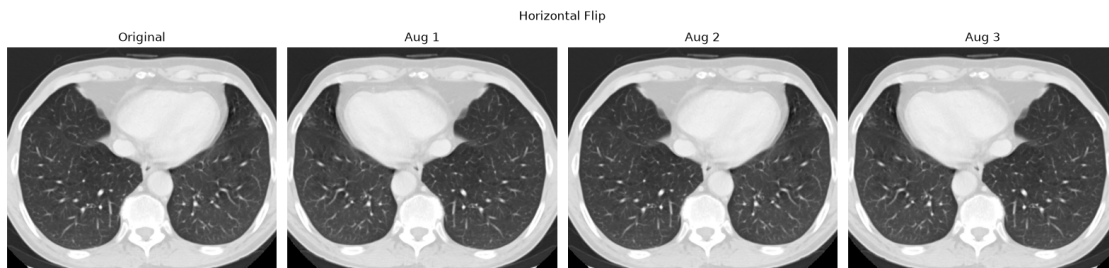
# Brightness
display_augmented_images(sample_path, ImageDataGenerator(brightness_range=[0.5, \
    ↪1.5]), title='Random Brightness')

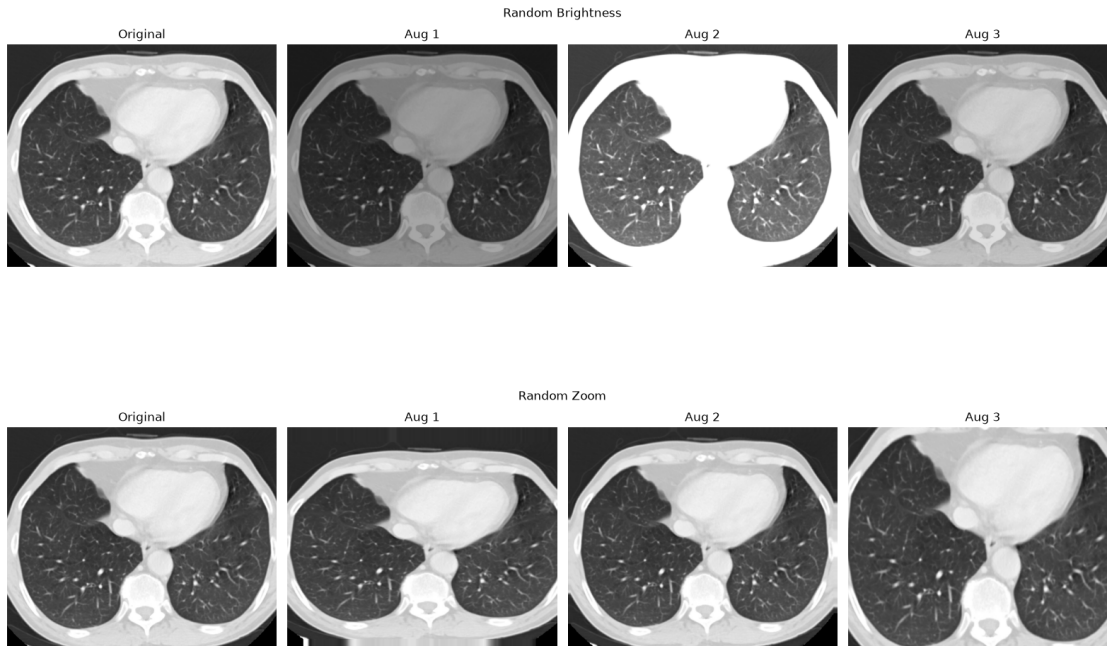
# Zoom
```

```
display_augmented_images(sample_path, ImageDataGenerator(zoom_range=0.3),  
↪title='Random Zoom')
```

2026-07-10 21:20:22.285104: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.

To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.





1.5 Split Dataset into Train / Validation / Test

```
[6]: import os, shutil
from sklearn.model_selection import train_test_split

original_dataset_dir = 'extracted_data/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 d in [train_dir, validation_dir, test_dir]:
    os.makedirs(d, exist_ok=True)

train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2

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)

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

# Print counts
for split in ['train', 'validation', 'test']:
    split_path = os.path.join(base_dir, split)
    for cls in os.listdir(split_path):
        count = len(os.listdir(os.path.join(split_path, cls)))
        print(f'{split}/{cls}: {count} images')

```

```

train/Bengin cases: 83 images
train/Normal cases: 291 images
train/Malignant cases: 392 images
validation/Bengin cases: 12 images
validation/Normal cases: 41 images
validation/Malignant cases: 56 images
test/Bengin cases: 25 images
test/Normal cases: 84 images
test/Malignant cases: 113 images

```

2 Lecture 2 — Training and Evaluating CNN Models (TensorFlow/Keras)

Prepared by Dr. Hager Saleh | Midocean University

```

[7]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
↳Dropout, GlobalAveragePooling2D
from tensorflow.keras.applications import ResNet50, EfficientNetB0, InceptionV3
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

print('TensorFlow version:', tf.__version__)

```

TensorFlow version: 2.16.2

2.1 Data Generators with Augmentation

```

[8]: IMG_SIZE = (240, 240)
    BATCH_SIZE = 32

def create_data_generators(train_dir, val_dir, test_dir, img_size=IMG_SIZE,
    ↪batch_size=BATCH_SIZE):
    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)

    train_generator = train_datagen.flow_from_directory(
        train_dir, target_size=img_size, batch_size=batch_size,
        class_mode='categorical', shuffle=True)
    val_generator = val_test_datagen.flow_from_directory(
        val_dir, target_size=img_size, batch_size=batch_size,
        class_mode='categorical', shuffle=False)
    test_generator = val_test_datagen.flow_from_directory(
        test_dir, target_size=img_size, batch_size=batch_size,
        class_mode='categorical', shuffle=False)

    return train_generator, val_generator, test_generator

train_gen, val_gen, test_gen = create_data_generators(
    'dataset/train', 'dataset/validation', 'dataset/test')

input_shape = (240, 240, 3)
num_classes = len(train_gen.class_indices)
print('Classes:', train_gen.class_indices)
print('Num classes:', num_classes)

```

Found 766 images belonging to 3 classes.

Found 109 images belonging to 3 classes.

Found 222 images belonging to 3 classes.

```
Classes: {'Benign cases': 0, 'Malignant cases': 1, 'Normal cases': 2}
Num classes: 3
```

2.2 Evaluation Function

```
[9]: def evaluate_model(model_name, y_true, y_pred_classes):
    labels = list(test_gen.class_indices.keys())
    accuracy = accuracy_score(y_true, y_pred_classes)
    precision = precision_score(y_true, y_pred_classes, average='weighted',
    ↪zero_division=0)
    recall = recall_score(y_true, y_pred_classes, average='weighted',
    ↪zero_division=0)
    f1 = f1_score(y_true, y_pred_classes, average='weighted',
    ↪zero_division=0)
    cm = confusion_matrix(y_true, y_pred_classes)

    print(f'\n===== {model_name} =====')
    print(f'Accuracy : {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall : {recall:.4f}')
    print(f'F1-score : {f1:.4f}')
    print('\nClassification Report:')
    print(classification_report(y_true, y_pred_classes, target_names=labels,
    ↪zero_division=0))

    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(f'Confusion Matrix - {model_name}')
    plt.tight_layout()
    plt.show()

def predict_and_evaluate(model, model_name):
    y_pred = model.predict(test_gen)
    y_true = test_gen.classes
    y_pred_classes = np.argmax(y_pred, axis=1)
    evaluate_model(model_name, y_true, y_pred_classes)
```

2.3 Model 1: Custom CNN

```
[10]: def build_cnn_model(input_shape, num_classes):
    model = Sequential([
        tf.keras.Input(shape=input_shape),
        Conv2D(32, (3, 3), activation='relu'),
        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

cnn_model = build_cnn_model(input_shape, num_classes)
cnn_model.summary()

history_cnn = cnn_model.fit(train_gen, epochs=10, validation_data=val_gen)
predict_and_evaluate(cnn_model, 'Custom CNN')

```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 238, 238, 32)	896
max_pooling2d (MaxPooling2D)	(None, 119, 119, 32)	0
conv2d_1 (Conv2D)	(None, 117, 117, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 58, 58, 64)	0
conv2d_2 (Conv2D)	(None, 56, 56, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 128)	0
flatten (Flatten)	(None, 100352)	0
dense (Dense)	(None, 128)	12,845,184
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 3)	387

Total params: 12,938,819 (49.36 MB)

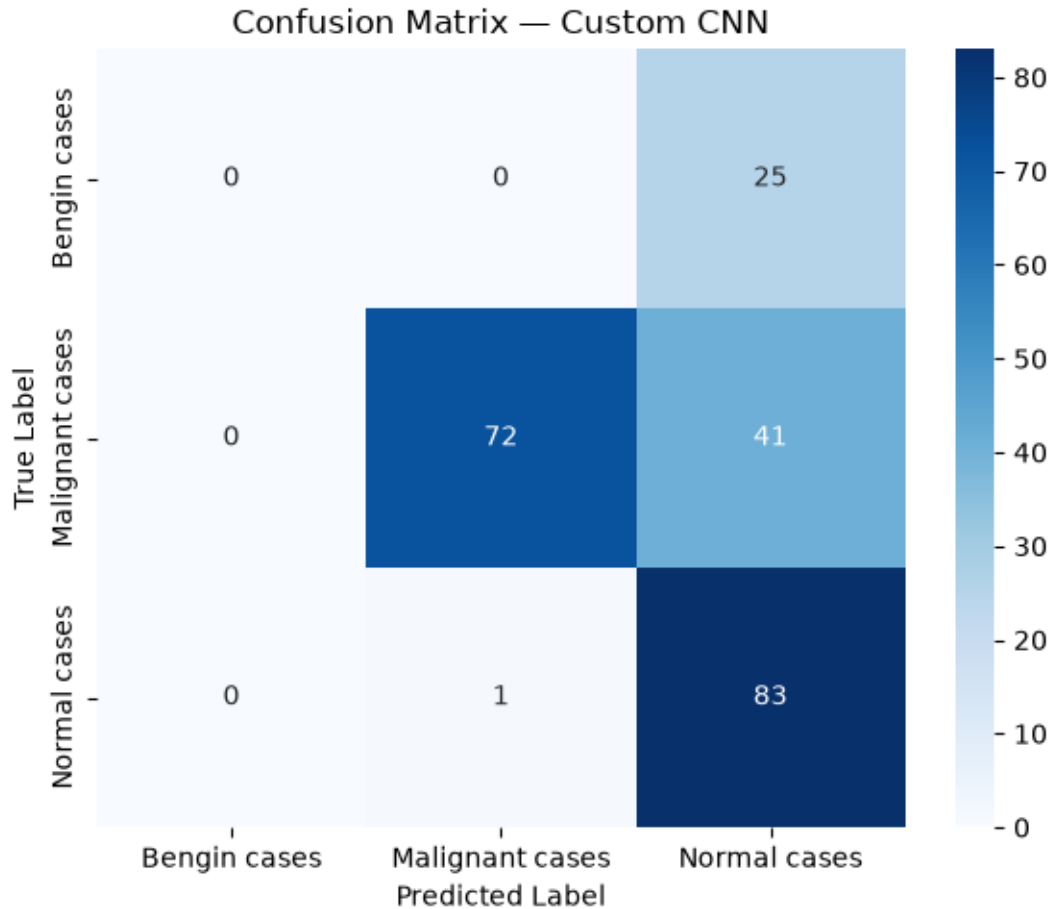
Trainable params: 12,938,819 (49.36 MB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10
24/24 34s 1s/step - accuracy: 0.4726 - loss: 1.1478 - val_accuracy: 0.3761 - val_loss: 0.9591
Epoch 2/10
24/24 32s 1s/step - accuracy: 0.4713 - loss: 0.9789 - val_accuracy: 0.5872 - val_loss: 0.8851
Epoch 3/10
24/24 41s 2s/step - accuracy: 0.5496 - loss: 0.9514 - val_accuracy: 0.5872 - val_loss: 0.8636
Epoch 4/10
24/24 38s 2s/step - accuracy: 0.5953 - loss: 0.9057 - val_accuracy: 0.5872 - val_loss: 0.8165
Epoch 5/10
24/24 30s 1s/step - accuracy: 0.5783 - loss: 0.9056 - val_accuracy: 0.7339 - val_loss: 0.7631
Epoch 6/10
24/24 31s 1s/step - accuracy: 0.5849 - loss: 0.8996 - val_accuracy: 0.6514 - val_loss: 0.7699
Epoch 7/10
24/24 31s 1s/step - accuracy: 0.6005 - loss: 0.8694 - val_accuracy: 0.7064 - val_loss: 0.6750
Epoch 8/10
24/24 32s 1s/step - accuracy: 0.6514 - loss: 0.8008 - val_accuracy: 0.8257 - val_loss: 0.5421
Epoch 9/10
24/24 30s 1s/step - accuracy: 0.6332 - loss: 0.8347 - val_accuracy: 0.6697 - val_loss: 0.7239
Epoch 10/10
24/24 31s 1s/step - accuracy: 0.6671 - loss: 0.7804 - val_accuracy: 0.7248 - val_loss: 0.6268
7/7 2s 309ms/step
===== Custom CNN =====
Accuracy : 0.6982
Precision: 0.7128
Recall : 0.6982
F1-score : 0.6636
Classification Report:

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.99	0.64	0.77	113
Normal cases	0.56	0.99	0.71	84
accuracy			0.70	222
macro avg	0.51	0.54	0.50	222

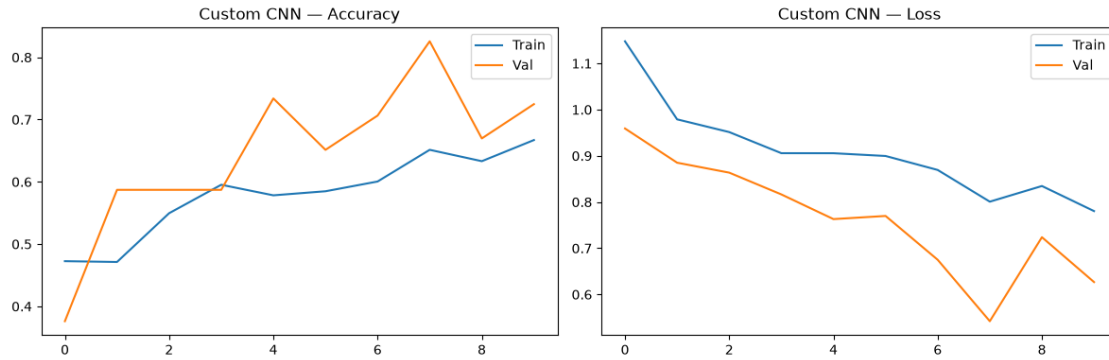
weighted avg 0.71 0.70 0.66 222



2.4 Plot Training History — CNN

```
[11]: def plot_history(history, model_name):  
    fig, axes = plt.subplots(1, 2, figsize=(12, 4))  
    axes[0].plot(history.history['accuracy'], label='Train')  
    axes[0].plot(history.history['val_accuracy'], label='Val')  
    axes[0].set_title(f'{model_name} - Accuracy')  
    axes[0].legend()  
    axes[1].plot(history.history['loss'], label='Train')  
    axes[1].plot(history.history['val_loss'], label='Val')  
    axes[1].set_title(f'{model_name} - Loss')  
    axes[1].legend()  
    plt.tight_layout()  
    plt.show()
```

```
plot_history(history_cnn, 'Custom CNN')
```



2.5 Model 2: ResNet50

```
[12]: def build_resnet50_model(input_shape, num_classes):
    base_model = ResNet50(weights='imagenet', include_top=False,
    ↪input_shape=input_shape)
    base_model.trainable = False
    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

resnet_model = build_resnet50_model(input_shape, num_classes)
history_resnet = resnet_model.fit(train_gen, epochs=5, validation_data=val_gen)
predict_and_evaluate(resnet_model, 'ResNet50')
plot_history(history_resnet, 'ResNet50')
```

Epoch 1/5

24/24 61s 2s/step - accuracy: 0.5144 - loss: 1.0463 - val_accuracy: 0.5138 - val_loss: 0.9707

Epoch 2/5

24/24 52s 2s/step - accuracy: 0.5209 - loss: 0.9567 - val_accuracy: 0.5229 - val_loss: 0.9387

Epoch 3/5

24/24 50s 2s/step - accuracy: 0.4935 - loss: 0.9650 - val_accuracy: 0.5138 - val_loss: 0.9367

Epoch 4/5

24/24 48s 2s/step - accuracy: 0.4817 - loss: 0.9769 - val_accuracy: 0.5138 - val_loss: 0.9407

Epoch 5/5
 24/24 48s 2s/step - accuracy: 0.4987 - loss: 0.9558 - val_accuracy: 0.5138 -
 val_loss: 0.9395

7/7 15s 2s/step

==== ResNet50 =====

Accuracy : 0.5090

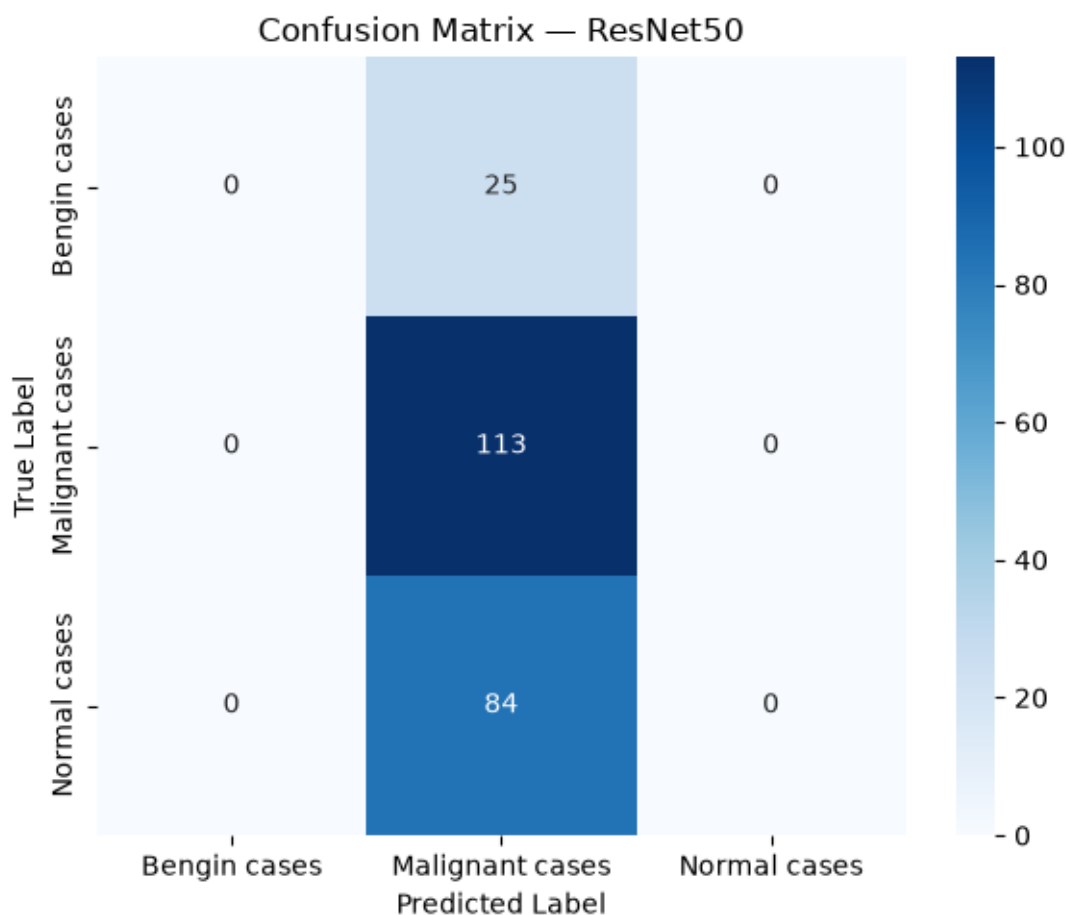
Precision: 0.2591

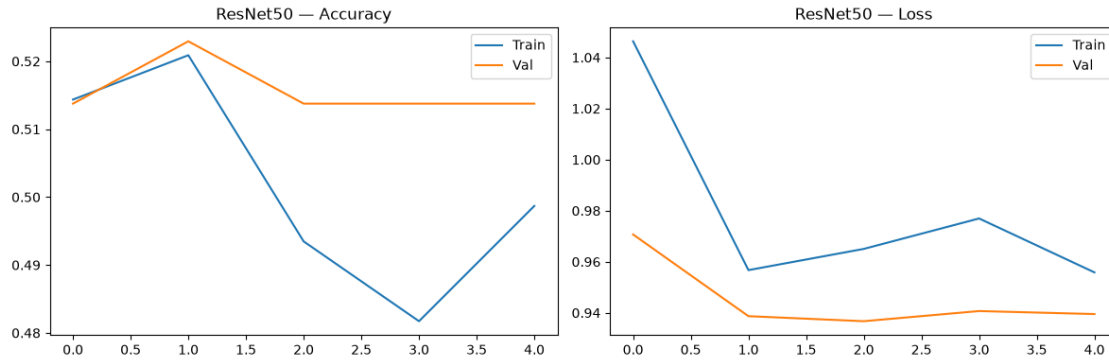
Recall : 0.5090

F1-score : 0.3434

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





2.6 Model 3: EfficientNetB0

```
[13]: def build_efficientnet_model(input_shape, num_classes):
    base_model = EfficientNetB0(weights='imagenet', include_top=False,
    ↪input_shape=input_shape)
    base_model.trainable = False
    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

efficientnet_model = build_efficientnet_model(input_shape, num_classes)
history_eff = efficientnet_model.fit(train_gen, epochs=5,
    ↪validation_data=val_gen)
predict_and_evaluate(efficientnet_model, 'EfficientNetB0')
plot_history(history_eff, 'EfficientNetB0')
```

Epoch 1/5

24/24 48s 2s/step - accuracy: 0.4687 - loss: 1.0260 - val_accuracy: 0.5138 - val_loss: 0.9644

Epoch 2/5

24/24 39s 2s/step - accuracy: 0.4883 - loss: 0.9841 - val_accuracy: 0.5138 - val_loss: 0.9557

Epoch 3/5

24/24 40s 2s/step - accuracy: 0.4935 - loss: 0.9761 - val_accuracy: 0.5138 - val_loss: 0.9588

Epoch 4/5

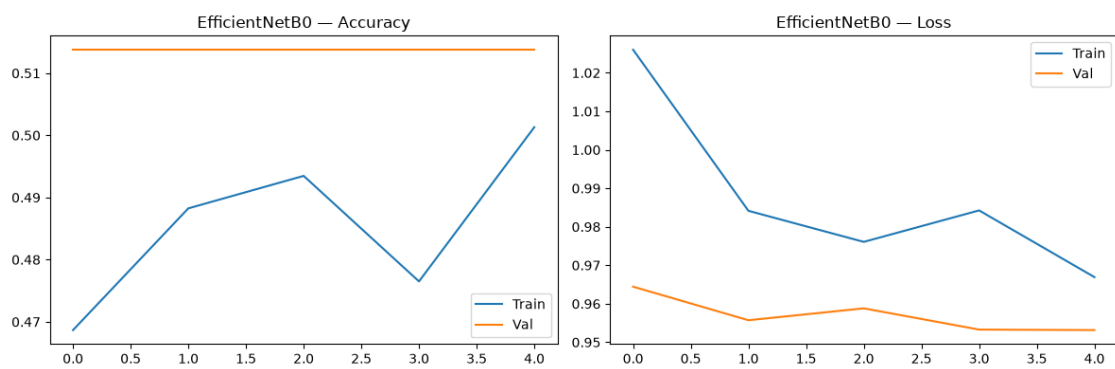
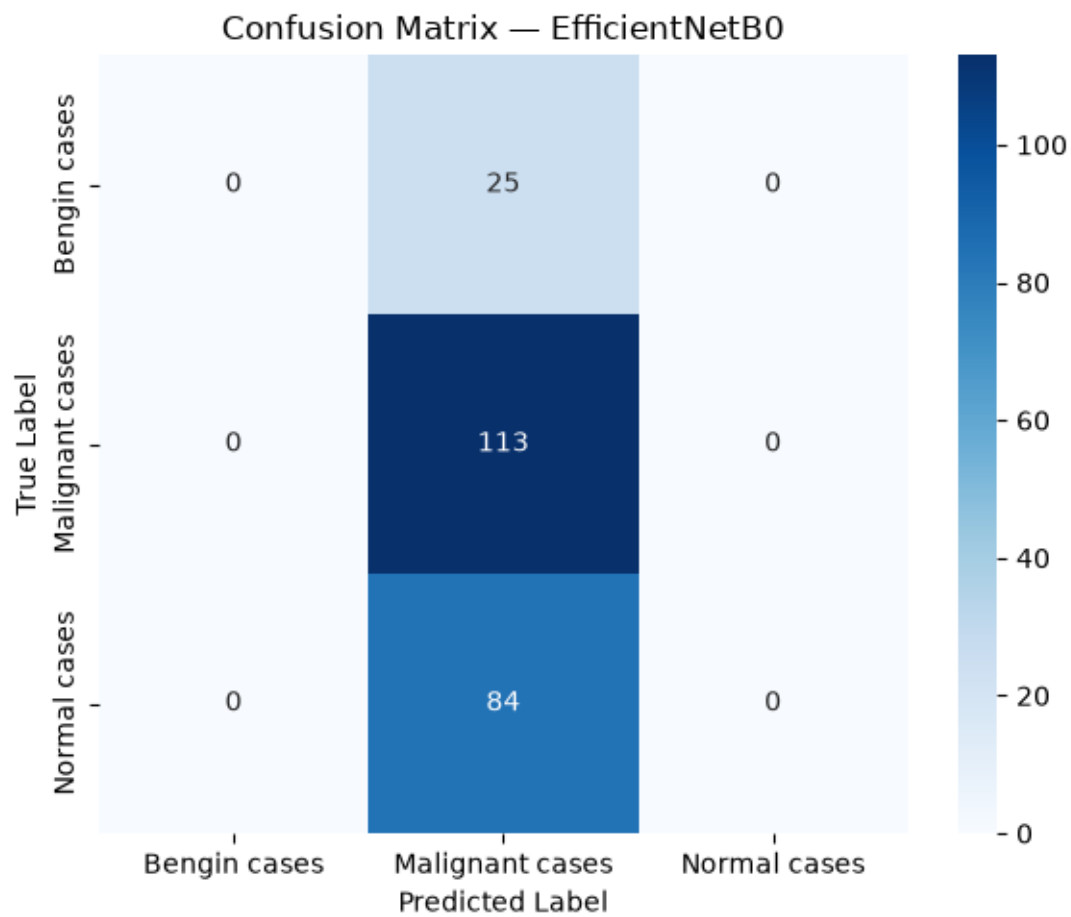
24/24 43s 2s/step - accuracy: 0.4765 - loss: 0.9842 - val_accuracy: 0.5138 -

```

val_loss: 0.9533
Epoch 5/5
24/24 42s 2s/step - accuracy: 0.5013 - loss: 0.9669 - val_accuracy: 0.5138 -
val_loss: 0.9532
WARNING:tensorflow:5 out of the last 15 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x13d546ca0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling\_retracing and
https://www.tensorflow.org/api\_docs/python/tf/function for more details.
7/7 13s 2s/step
===== EfficientNetB0 =====
Accuracy : 0.5090
Precision: 0.2591
Recall    : 0.5090
F1-score  : 0.3434
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



2.7 Model 4: InceptionV3

```
[14]: # InceptionV3 requires minimum 75x75 input - re-create generators at 240x240
      ↪(already set)
def build_inceptionv3_model(input_shape, num_classes):
    base_model = InceptionV3(weights='imagenet', include_top=False,
    ↪input_shape=input_shape)
    base_model.trainable = False
    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

inception_model = build_inceptionv3_model(input_shape, num_classes)
history_inc = inception_model.fit(train_gen, epochs=5, validation_data=val_gen)
predict_and_evaluate(inception_model, 'InceptionV3')
plot_history(history_inc, 'InceptionV3')
```

Epoch 1/5

24/24 34s 1s/step - accuracy: 0.6527 - loss: 1.0512 - val_accuracy: 0.8624 -
val_loss: 0.4516

Epoch 2/5

24/24 27s 1s/step - accuracy: 0.8185 - loss: 0.5048 - val_accuracy: 0.8624 -
val_loss: 0.3868

Epoch 3/5

24/24 28s 1s/step - accuracy: 0.8342 - loss: 0.4776 - val_accuracy: 0.8532 -
val_loss: 0.3598

Epoch 4/5

24/24 31s 1s/step - accuracy: 0.8251 - loss: 0.4666 - val_accuracy: 0.8624 -
val_loss: 0.3549

Epoch 5/5

24/24 34s 1s/step - accuracy: 0.8446 - loss: 0.4343 - val_accuracy: 0.8624 -
val_loss: 0.3314

WARNING:tensorflow:5 out of the last 15 calls to <function

TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x13f6777e0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and

https://www.tensorflow.org/api_docs/python/tf/function for more details.

7/7 13s 1s/step

===== InceptionV3 =====

Accuracy : 0.8784

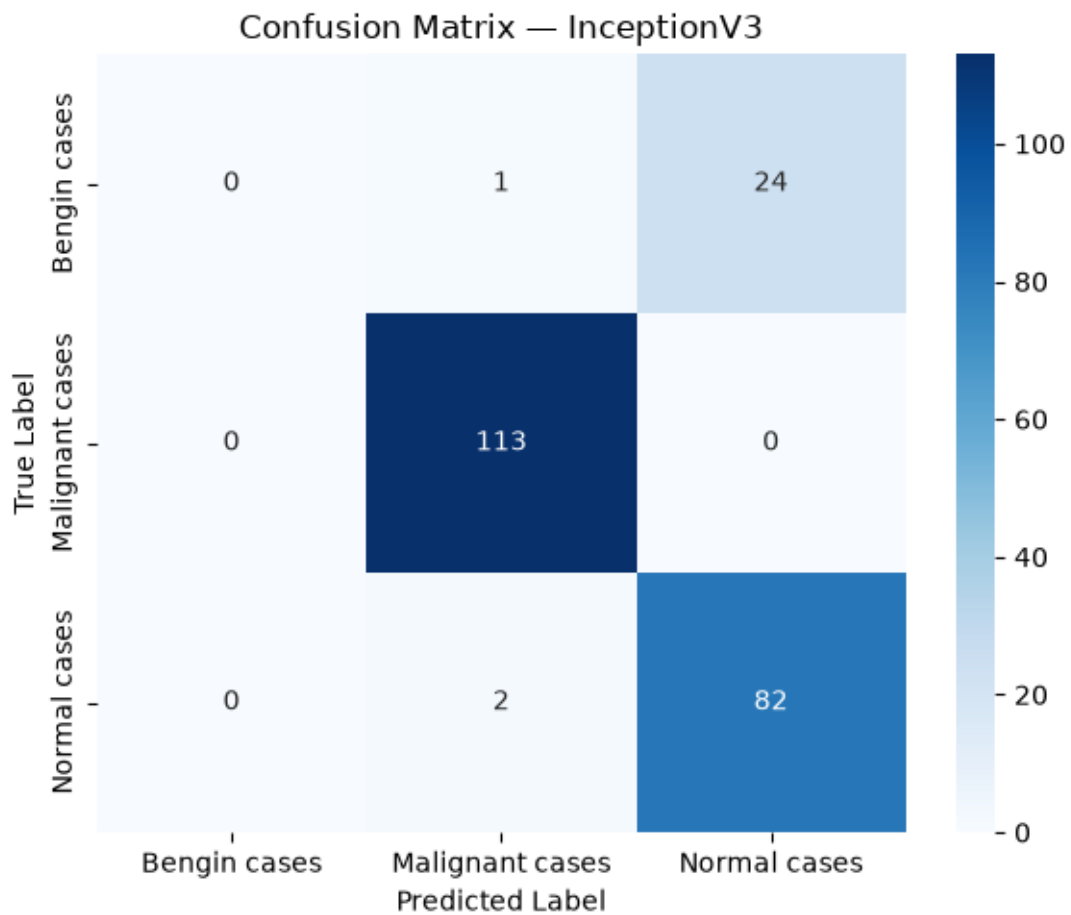
Precision: 0.7886

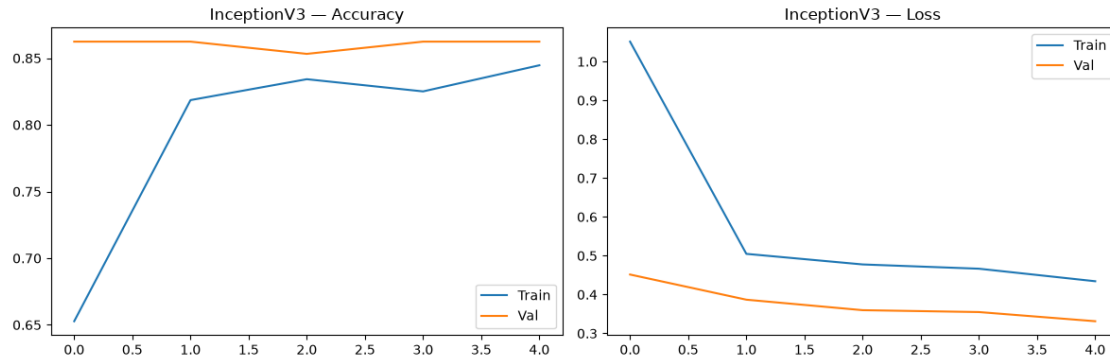
Recall : 0.8784

F1-score : 0.8289

Classification Report:

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.97	1.00	0.99	113
Normal cases	0.77	0.98	0.86	84
accuracy			0.88	222
macro avg	0.58	0.66	0.62	222
weighted avg	0.79	0.88	0.83	222





3 Lecture 3 — Training and Evaluating CNN Models Using Py-Torch

Prepared by Dr. Hager Saleh | Midocean University

```
[15]: 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 roc_curve, auc

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)
```

Using device: cpu

3.1 Get Class Names

```
[16]: def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes

classes = get_classes('dataset/train')
num_classes_pt = len(classes)
print('Classes:', classes)
print('Num classes:', num_classes_pt)
```

Classes: ['Benign cases', 'Malignant cases', 'Normal cases']
Num classes: 3

3.2 Data Loaders with PyTorch Transforms

```
[17]: def get_data_loaders(data_dir, batch_size, train=False):
    if train:
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
            T.RandomRotation(10),
            T.Resize((299, 299)),
            T.ToTensor(),
            T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ])
        train_data = datasets.ImageFolder(os.path.join(data_dir, 'train'),
        ↪transform=transform)
        train_loader = DataLoader(train_data, batch_size=batch_size,
        ↪shuffle=True, num_workers=0)
        return train_loader, len(train_data)
    else:
        transform = T.Compose([
            T.Resize((299, 299)),
            T.ToTensor(),
            T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ])
        val_data = datasets.ImageFolder(os.path.join(data_dir, 'validation'),
        ↪transform=transform)
        test_data = datasets.ImageFolder(os.path.join(data_dir, 'test'),
        ↪transform=transform)
        val_loader = DataLoader(val_data, batch_size=batch_size,
        ↪shuffle=False, num_workers=0)
        test_loader = DataLoader(test_data, batch_size=batch_size,
        ↪shuffle=False, num_workers=0)
        return val_loader, test_loader, len(val_data), len(test_data)

BATCH_SIZE_PT = 32
DATASET_PATH = 'dataset'

train_loader_pt, train_len = get_data_loaders(DATASET_PATH, BATCH_SIZE_PT,
        ↪train=True)
val_loader_pt, test_loader_pt, val_len, test_len =
        ↪get_data_loaders(DATASET_PATH, BATCH_SIZE_PT, train=False)

dataloaders_pt = {'train': train_loader_pt, 'val': val_loader_pt}
dataset_sizes_pt = {'train': train_len, 'val': val_len}
print(f'Train: {train_len} | Val: {val_len} | Test: {test_len}')
```

Train: 766 | Val: 109 | Test: 222

3.3 Training Loop

```
[18]: def train_model_pt(model, criterion, optimizer, scheduler, num_epochs=5):
    for epoch in range(num_epochs):
        print(f'Epoch {epoch+1}/{num_epochs} {"-"*10}')
        for phase in ['train', 'val']:
            model.train() if phase == 'train' else model.eval()
            running_loss = 0.0
            running_corrects = 0
            for inputs, labels in dataloaders_pt[phase]:
                inputs, labels = inputs.to(device), labels.to(device)
                optimizer.zero_grad()
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    if isinstance(outputs, tuple) or hasattr(outputs, 'logits'):
                        main_out = outputs.logits if hasattr(outputs, 'logits')
                    else outputs[0]
                        aux_out = outputs.aux_logits if hasattr(outputs,
                    'aux_logits') else None
                        loss = criterion(main_out, labels)
                        if aux_out is not None:
                            loss += 0.4 * criterion(aux_out, labels)
                        else:
                            main_out = outputs
                            loss = criterion(main_out, labels)
                        _, preds = torch.max(main_out, 1)
                        if phase == 'train':
                            loss.backward()
                            optimizer.step()
                        running_loss += loss.item() * inputs.size(0)
                        running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()
            epoch_loss = running_loss / dataset_sizes_pt[phase]
            epoch_acc = running_corrects.double() / dataset_sizes_pt[phase]
            print(f' {phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

    return model

def collect_predictions(model):
    y_true, y_pred, y_proba = [], [], []
    model.eval()
    with torch.no_grad():
        for x, y in tqdm(test_loader_pt):
            out = model(x.to(device))
            prob = torch.softmax(out, dim=1)
            y_true.extend(y.numpy())
            y_proba.extend(prob.cpu().numpy())
```

```

        y_pred.extend(torch.argmax(prob, dim=1).cpu().numpy())
    return np.array(y_true), np.array(y_pred), np.array(y_proba)

def evaluate_pt(model_name, y_true, y_pred, classes):
    acc = accuracy_score(y_true, y_pred)
    pre = precision_score(y_true, y_pred, average='weighted', zero_division=0)
    rec = recall_score(y_true, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)
    cm = confusion_matrix(y_true, y_pred)
    print(f'\n===== {model_name} =====')
    print(f'Accuracy : {acc:.4f} Precision: {pre:.4f} Recall: {rec:.4f} F1: {f1:.4f}')
    print(classification_report(y_true, y_pred, target_names=classes,
    zero_division=0))
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=classes, yticklabels=classes)
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted'); plt.ylabel('True')
    plt.tight_layout(); plt.show()

```

3.4 PyTorch Model 1: EfficientNetB0 (Pretrained)

```

[19]: model_eff_pt = models.efficientnet_b0(pretrained=True)
model_eff_pt.classifier[1] = nn.Linear(model_eff_pt.classifier[1].in_features,
    num_classes_pt)
model_eff_pt = model_eff_pt.to(device)

criterion_pt = nn.CrossEntropyLoss().to(device)
optimizer_eff = optim.AdamW(model_eff_pt.parameters(), lr=0.001)
scheduler_eff = optim.lr_scheduler.StepLR(optimizer_eff, step_size=3, gamma=0.
    97)

model_eff_pt = train_model_pt(model_eff_pt, criterion_pt, optimizer_eff,
    scheduler_eff, num_epochs=5)
y_true_pt, y_pred_pt, y_proba_pt = collect_predictions(model_eff_pt)
evaluate_pt('EfficientNetB0 (pretrained)', y_true_pt, y_pred_pt, classes)

```

```

Epoch 1/5 -----
  train Loss: 0.4631  Acc: 0.8133
  val Loss: 0.2592  Acc: 0.8807
Epoch 2/5 -----
  train Loss: 0.2817  Acc: 0.8851
  val Loss: 0.2801  Acc: 0.8807
Epoch 3/5 -----
  train Loss: 0.2397  Acc: 0.8956
  val Loss: 0.5021  Acc: 0.8899

```

Epoch 4/5 -----

train Loss: 0.2112 Acc: 0.9138

val Loss: 0.3826 Acc: 0.8899

Epoch 5/5 -----

train Loss: 0.1903 Acc: 0.9191

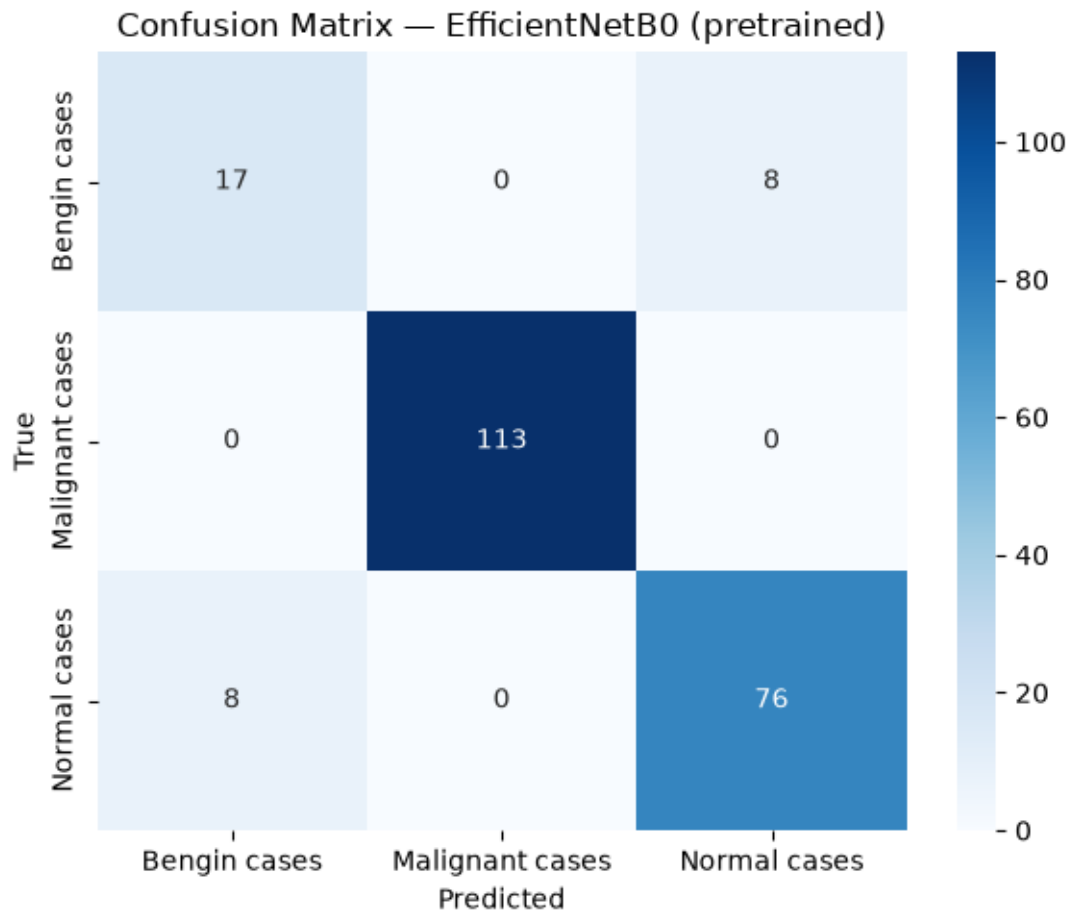
val Loss: 0.1278 Acc: 0.9633

==== EfficientNetB0 (pretrained) =====

Accuracy : 0.9279 Precision: 0.9279 Recall: 0.9279 F1: 0.9279

	precision	recall	f1-score	support
Bengin cases	0.68	0.68	0.68	25
Malignant cases	1.00	1.00	1.00	113
Normal cases	0.90	0.90	0.90	84
accuracy			0.93	222
macro avg	0.86	0.86	0.86	222
weighted avg	0.93	0.93	0.93	222

100%|| 7/7 [00:12<00:00, 1.81s/it]



3.5 PyTorch Model 2: ResNet18 (No Pretrained)

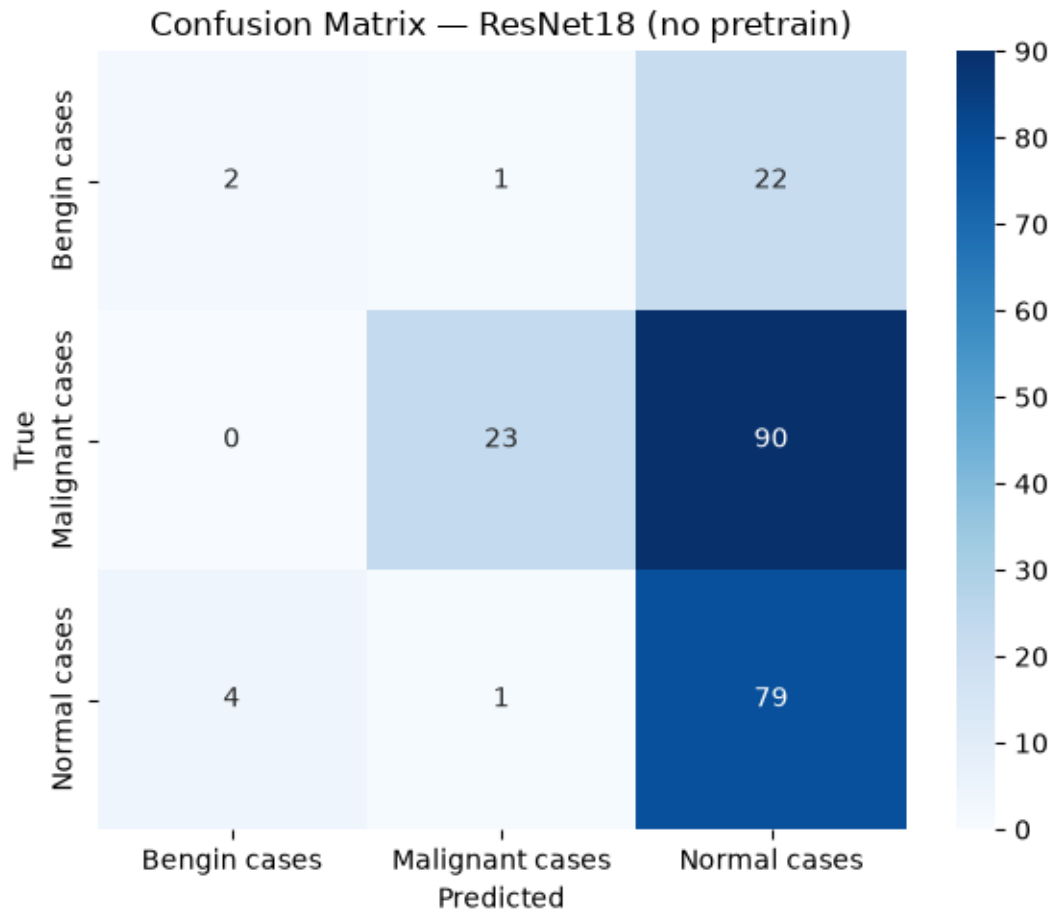
```
[20]: model_res_pt = models.resnet18(pretrained=False)
model_res_pt.fc = nn.Linear(model_res_pt.fc.in_features, num_classes_pt)
model_res_pt = model_res_pt.to(device)

optimizer_res = optim.AdamW(model_res_pt.parameters(), lr=0.001)
scheduler_res = optim.lr_scheduler.StepLR(optimizer_res, step_size=3, gamma=0.
↪97)

model_res_pt = train_model_pt(model_res_pt, criterion_pt, optimizer_res, ↪
↪scheduler_res, num_epochs=5)
y_true_res, y_pred_res, _ = collect_predictions(model_res_pt)
evaluate_pt('ResNet18 (no pretrain)', y_true_res, y_pred_res, classes)
```

```
Epoch 1/5 -----
  train Loss: 0.7515  Acc: 0.6958
  val Loss: 18.1396  Acc: 0.1101
Epoch 2/5 -----
  train Loss: 0.6098  Acc: 0.7598
  val Loss: 8.3047  Acc: 0.5138
Epoch 3/5 -----
  train Loss: 0.4916  Acc: 0.8251
  val Loss: 2.0710  Acc: 0.5780
Epoch 4/5 -----
  train Loss: 0.5313  Acc: 0.7898
  val Loss: 3.9595  Acc: 0.3761
Epoch 5/5 -----
  train Loss: 0.4631  Acc: 0.8225
  val Loss: 1.2501  Acc: 0.4679
===== ResNet18 (no pretrain) =====
Accuracy : 0.4685  Precision: 0.6623  Recall: 0.4685  F1: 0.4016
      precision    recall  f1-score   support
  Bengin cases      0.33      0.08      0.13         25
Malignant cases      0.92      0.20      0.33        113
  Normal cases      0.41      0.94      0.57         84
    accuracy
    macro avg      0.56      0.41      0.35        222
  weighted avg      0.66      0.47      0.40        222

100%|| 7/7 [00:08<00:00, 1.25s/it]
```



3.6 PyTorch Model 3: DenseNet121 (No Pretrained)

```
[21]: model_den_pt = models.densenet121(pretrained=False)
model_den_pt.classifier = nn.Linear(model_den_pt.classifier.in_features,
    ↪ num_classes_pt)
model_den_pt = model_den_pt.to(device)

optimizer_den = optim.AdamW(model_den_pt.parameters(), lr=0.001)
scheduler_den = optim.lr_scheduler.StepLR(optimizer_den, step_size=3, gamma=0.
    ↪ 97)

model_den_pt = train_model_pt(model_den_pt, criterion_pt, optimizer_den,
    ↪ scheduler_den, num_epochs=3)
y_true_den, y_pred_den, _ = collect_predictions(model_den_pt)
evaluate_pt('DenseNet121 (no pretrain)', y_true_den, y_pred_den, classes)
```

```
Epoch 1/3 -----
train Loss: 0.5976 Acc: 0.7559
```

```

val Loss: 1.0238  Acc: 0.6239
Epoch 2/3  -----
train Loss: 0.4129  Acc: 0.8512
val Loss: 2.5257  Acc: 0.5872
Epoch 3/3  -----
train Loss: 0.4397  Acc: 0.8277
val Loss: 1.2329  Acc: 0.5596
===== DenseNet121 (no pretrain) =====
Accuracy : 0.5180  Precision: 0.7066  Recall: 0.5180  F1: 0.5006

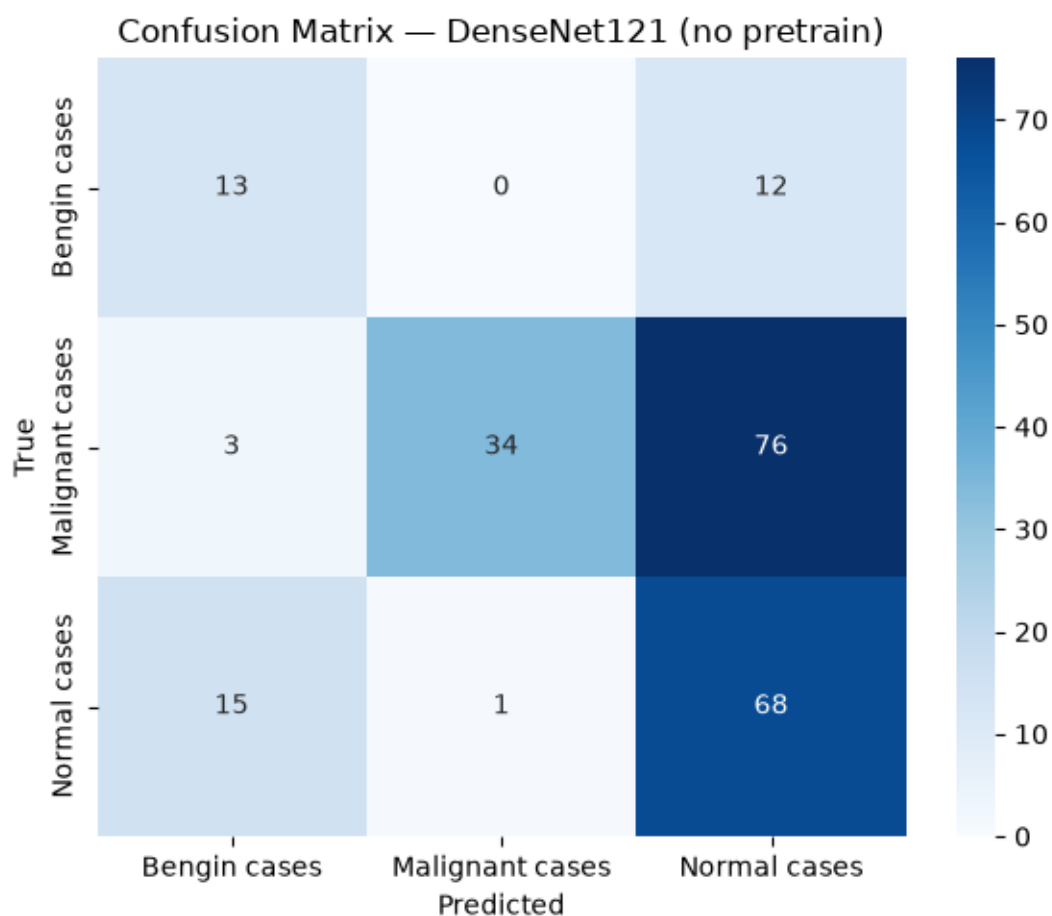
```

	precision	recall	f1-score	support
Bengin cases	0.42	0.52	0.46	25
Malignant cases	0.97	0.30	0.46	113
Normal cases	0.44	0.81	0.57	84
accuracy			0.52	222
macro avg	0.61	0.54	0.50	222
weighted avg	0.71	0.52	0.50	222

```

100%|| 7/7 [00:30<00:00, 4.38s/it]

```



4 Lecture 4 — Reducing Image Size and Training Models

Prepared by Dr. Hager Saleh | Midocean University

4.1 Reduce Image Size with PIL

```
[22]: from PIL import Image as PILImage

input_root = 'extracted_data/The IQ-OTHNCCD lung cancer dataset'
output_root = 'dataset_resized'
resize_size = (256, 256)

os.makedirs(output_root, exist_ok=True)
total_processed = 0

for class_name in os.listdir(input_root):
    class_path = os.path.join(input_root, class_name)
    if not os.path.isdir(class_path):
        continue
    output_class_path = os.path.join(output_root, class_name)
    os.makedirs(output_class_path, exist_ok=True)
    for image_name in os.listdir(class_path):
        if image_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
            in_path = os.path.join(class_path, image_name)
            out_name = os.path.splitext(image_name)[0] + '.jpg'
            out_path = os.path.join(output_class_path, out_name)
            try:
                with PILImage.open(in_path) as img:
                    img = img.convert('RGB')
                    img = img.resize(resize_size)
                    img.save(out_path, 'JPEG')
                total_processed += 1
            except Exception as e:
                print(f'Failed: {in_path}: {e}')

print(f'Done! Processed {total_processed} images -> {output_root}/')
```

Done! Processed 1097 images -> dataset_resized/

4.2 Compare Original vs Resized Size

```
[23]: import subprocess

def folder_size_mb(path):
```

```

total = 0
for dirpath, _, filenames in os.walk(path):
    for f in filenames:
        total += os.path.getsize(os.path.join(dirpath, f))
return total / (1024 * 1024)

orig_mb = folder_size_mb(input_root)
resized_mb = folder_size_mb(output_root)
print(f'Original dataset : {orig_mb:.1f} MB')
print(f'Resized dataset : {resized_mb:.1f} MB')
print(f'Reduction : {(1 - resized_mb/orig_mb)*100:.1f}%')

```

```

Original dataset : 150.3 MB
Resized dataset : 10.5 MB
Reduction : 93.0%

```

4.3 Split Resized Dataset

```

[24]: base_dir_resized = 'dataset_resized_split'

for split in ['train', 'validation', 'test']:
    os.makedirs(os.path.join(base_dir_resized, split), exist_ok=True)

train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2

for class_name in os.listdir(output_root):
    class_path = os.path.join(output_root, class_name)
    if not os.path.isdir(class_path):
        continue
    split_and_copy_files(class_path, base_dir_resized)

for split in ['train', 'validation', 'test']:
    split_path = os.path.join(base_dir_resized, split)
    for cls in sorted(os.listdir(split_path)):
        count = len(os.listdir(os.path.join(split_path, cls)))
        print(f'{split}/{cls}: {count} images')

```

```

train/Bengin cases: 83 images
train/Malignant cases: 392 images
train/Normal cases: 291 images
validation/Bengin cases: 12 images
validation/Malignant cases: 56 images
validation/Normal cases: 41 images
test/Bengin cases: 25 images
test/Malignant cases: 113 images
test/Normal cases: 84 images

```

4.4 Train on Resized Dataset — PyTorch DataLoaders

```
[25]: train_loader_r, train_len_r = get_data_loaders(base_dir_resized, BATCH_SIZE_PT,
    ↪train=True)
val_loader_r, test_loader_r, val_len_r, test_len_r =
    ↪get_data_loaders(base_dir_resized, BATCH_SIZE_PT, train=False)

dataloaders_pt = {'train': train_loader_r, 'val': val_loader_r}
dataset_sizes_pt = {'train': train_len_r, 'val': val_len_r}

classes_r = get_classes(os.path.join(base_dir_resized, 'train'))
num_classes_r = len(classes_r)
print('Classes:', classes_r)

# Redefine collect_predictions to use resized test loader
def collect_predictions_r(model):
    y_true, y_pred, y_proba = [], [], []
    model.eval()
    with torch.no_grad():
        for x, y in tqdm(test_loader_r):
            out = model(x.to(device))
            prob = torch.softmax(out, dim=1)
            y_true.extend(y.numpy())
            y_proba.extend(prob.cpu().numpy())
            y_pred.extend(torch.argmax(prob, dim=1).cpu().numpy())
    return np.array(y_true), np.array(y_pred), np.array(y_proba)
```

Classes: ['Bengin cases', 'Malignant cases', 'Normal cases']

4.5 Lecture 4 Model 1: EfficientNetB0 (Pretrained) on Resized Data

```
[26]: model_eff_r = models.efficientnet_b0(pretrained=True)
model_eff_r.classifier[1] = nn.Linear(model_eff_r.classifier[1].in_features,
    ↪num_classes_r)
model_eff_r = model_eff_r.to(device)

opt_r = optim.AdamW(model_eff_r.parameters(), lr=0.001)
sch_r = optim.lr_scheduler.StepLR(opt_r, step_size=3, gamma=0.97)

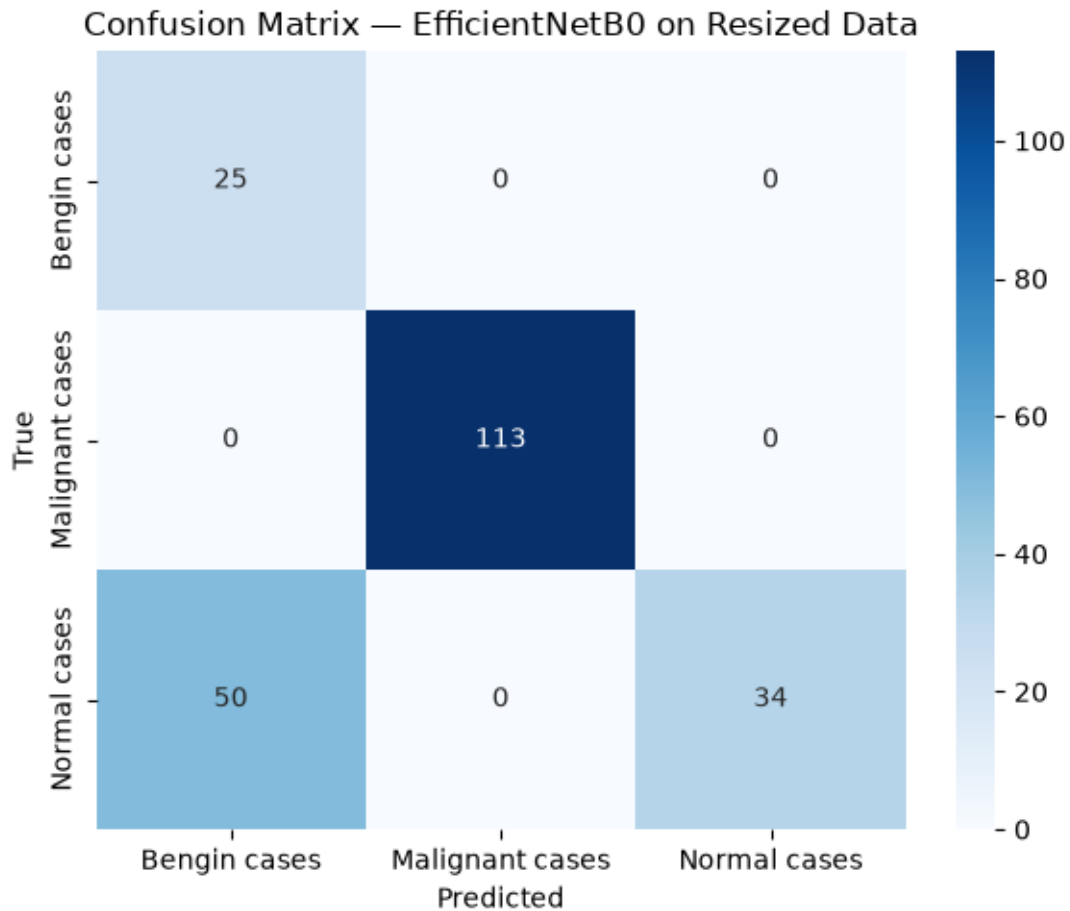
model_eff_r = train_model_pt(model_eff_r, criterion_pt, opt_r, sch_r,
    ↪num_epochs=5)
yt_r, yp_r, yprob_r = collect_predictions_r(model_eff_r)
evaluate_pt('EfficientNetB0 on Resized Data', yt_r, yp_r, classes_r)
```

```
Epoch 1/5 -----
  train Loss: 0.4594  Acc: 0.8159
  val Loss: 0.2987  Acc: 0.8349
Epoch 2/5 -----
```

```

train Loss: 0.2752 Acc: 0.8799
val Loss: 0.1978 Acc: 0.9083
Epoch 3/5 -----
train Loss: 0.2406 Acc: 0.8995
val Loss: 0.4068 Acc: 0.8257
Epoch 4/5 -----
train Loss: 0.1824 Acc: 0.9282
val Loss: 0.2292 Acc: 0.9083
Epoch 5/5 -----
train Loss: 0.1192 Acc: 0.9595
val Loss: 0.5445 Acc: 0.7982
===== EfficientNetB0 on Resized Data =====
Accuracy : 0.7748 Precision: 0.9249 Recall: 0.7748 F1: 0.7834
      precision    recall  f1-score   support
Bengin cases      0.33      1.00      0.50        25
Malignant cases    1.00      1.00      1.00       113
Normal cases       1.00      0.40      0.58        84
    accuracy
    macro avg      0.78      0.80      0.69       222
    weighted avg    0.92      0.77      0.78       222
100%|| 7/7 [00:11<00:00, 1.69s/it]

```



4.6 Lecture 4 Model 2: ResNet18 (No Pretrained) on Resized Data

```
[27]: model_res_r = models.resnet18(pretrained=False)
model_res_r.fc = nn.Linear(model_res_r.fc.in_features, num_classes_r)
model_res_r = model_res_r.to(device)

opt_res_r = optim.AdamW(model_res_r.parameters(), lr=0.001)
sch_res_r = optim.lr_scheduler.StepLR(opt_res_r, step_size=3, gamma=0.97)

model_res_r = train_model_pt(model_res_r, criterion_pt, opt_res_r, sch_res_r,
    ↪ num_epochs=3)
yt_res_r, yp_res_r, _ = collect_predictions_r(model_res_r)
evaluate_pt('ResNet18 on Resized Data', yt_res_r, yp_res_r, classes_r)
```

```
Epoch 1/3 -----
  train Loss: 0.8141  Acc: 0.6527
    val Loss: 13.3261  Acc: 0.1101
Epoch 2/3 -----
```

```

train Loss: 0.6200  Acc: 0.7611
val Loss: 0.5065  Acc: 0.8073
Epoch 3/3  -----
train Loss: 0.5170  Acc: 0.7885
val Loss: 1.5992  Acc: 0.5321
===== ResNet18 on Resized Data =====
Accuracy : 0.5495  Precision: 0.6478  Recall: 0.5495  F1: 0.4293

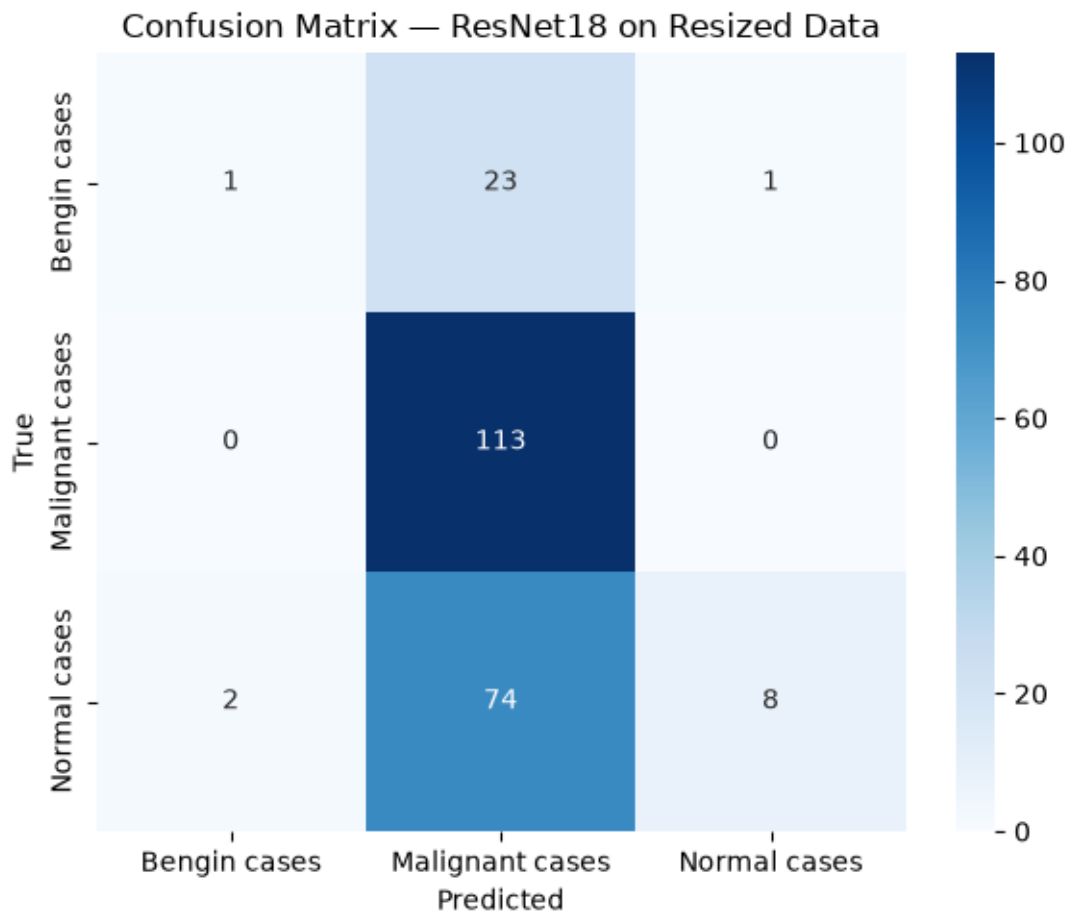
```

	precision	recall	f1-score	support
Bengin cases	0.33	0.04	0.07	25
Malignant cases	0.54	1.00	0.70	113
Normal cases	0.89	0.10	0.17	84
accuracy			0.55	222
macro avg	0.59	0.38	0.31	222
weighted avg	0.65	0.55	0.43	222

```

100%|| 7/7 [00:07<00:00, 1.14s/it]

```



4.7 Lecture 4 Model 3: DenseNet121 on Resized Data

```
[28]: model_den_r = models.densenet121(pretrained=False)
model_den_r.classifier = nn.Linear(model_den_r.classifier.in_features,
    ↪ num_classes_r)
model_den_r = model_den_r.to(device)

opt_den_r = optim.AdamW(model_den_r.parameters(), lr=0.001)
sch_den_r = optim.lr_scheduler.StepLR(opt_den_r, step_size=3, gamma=0.97)

model_den_r = train_model_pt(model_den_r, criterion_pt, opt_den_r, sch_den_r,
    ↪ num_epochs=3)
yt_den_r, yp_den_r, _ = collect_predictions_r(model_den_r)
evaluate_pt('DenseNet121 on Resized Data', yt_den_r, yp_den_r, classes_r)
```

Epoch 1/3 -----

train Loss: 0.7157 Acc: 0.6893

val Loss: 0.5845 Acc: 0.8440

Epoch 2/3 -----

train Loss: 0.5208 Acc: 0.8042

val Loss: 0.8413 Acc: 0.6697

Epoch 3/3 -----

train Loss: 0.4265 Acc: 0.8303

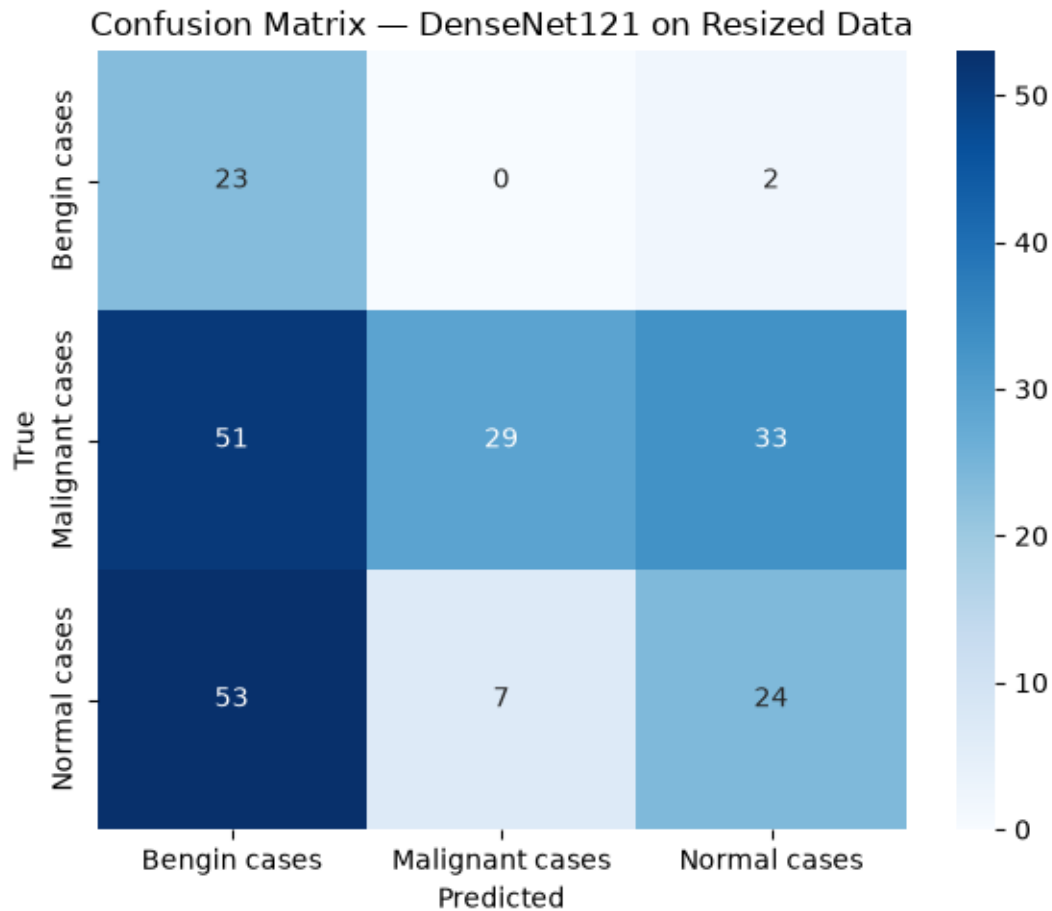
val Loss: 1.6415 Acc: 0.3761

==== DenseNet121 on Resized Data =====

Accuracy : 0.3423 Precision: 0.5843 Recall: 0.3423 F1: 0.3592

	precision	recall	f1-score	support
Bengin cases	0.18	0.92	0.30	25
Malignant cases	0.81	0.26	0.39	113
Normal cases	0.41	0.29	0.34	84
accuracy			0.34	222
macro avg	0.46	0.49	0.34	222
weighted avg	0.58	0.34	0.36	222

100%|| 7/7 [00:28<00:00, 4.14s/it]



4.8 Final Model Comparison Summary

```
[29]: import pandas as pd

results = {
    'Model': [
        'Custom CNN (Keras)',
        'ResNet50 (Keras, pretrained)',
        'EfficientNetB0 (Keras, pretrained)',
        'InceptionV3 (Keras, pretrained)',
        'EfficientNetB0 (PyTorch, pretrained)',
        'ResNet18 (PyTorch, no pretrain)',
        'DenseNet121 (PyTorch, no pretrain)',
        'EfficientNetB0 (Resized, pretrained)',
        'ResNet18 (Resized, no pretrain)',
        'DenseNet121 (Resized, no pretrain)',
    ],
    'Framework': ['Keras']*4 + ['PyTorch']*3 + ['PyTorch (Lec4)']*3,
```

```

    'Accuracy': [
        accuracy_score(test_gen.classes, np.argmax(cnn_model.predict(test_gen),
↪axis=1)),
        accuracy_score(test_gen.classes, np.argmax(resnet_model.
↪predict(test_gen), axis=1)),
        accuracy_score(test_gen.classes, np.argmax(efficientnet_model.
↪predict(test_gen), axis=1)),
        accuracy_score(test_gen.classes, np.argmax(inception_model.
↪predict(test_gen), axis=1)),
        accuracy_score(y_true_pt, y_pred_pt),
        accuracy_score(y_true_res, y_pred_res),
        accuracy_score(y_true_den, y_pred_den),
        accuracy_score(yt_r, yp_r),
        accuracy_score(yt_res_r, yp_res_r),
        accuracy_score(yt_den_r, yp_den_r),
    ]
}

df = pd.DataFrame(results)
df['Accuracy'] = df['Accuracy'].map(lambda x: f'{x*100:.2f}%')
print(df.to_string(index=False))

```

```

7/7 2s 208ms/step
7/7 11s 1s/step
7/7 9s 1s/step
7/7 6s 853ms/step

```

	Model	Framework	Accuracy
	Custom CNN (Keras)	Keras	69.82%
	ResNet50 (Keras, pretrained)	Keras	50.90%
	EfficientNetB0 (Keras, pretrained)	Keras	50.90%
	InceptionV3 (Keras, pretrained)	Keras	87.84%
	EfficientNetB0 (PyTorch, pretrained)	PyTorch	92.79%
	ResNet18 (PyTorch, no pretrain)	PyTorch	46.85%
	DenseNet121 (PyTorch, no pretrain)	PyTorch	51.80%
	EfficientNetB0 (Resized, pretrained)	PyTorch (Lec4)	77.48%
	ResNet18 (Resized, no pretrain)	PyTorch (Lec4)	54.95%
	DenseNet121 (Resized, no pretrain)	PyTorch (Lec4)	34.23%