



Practical Image Processing and Natural Language Processing Second Trimester 2026

Project 1

Prepared By

Name: Sultan Ali Senan ID No.: 253002563

At

College of Informatics

Midocean University

Project 1 — Lung Cancer Image Classification using CNN Models

Dataset: IQ-OTHNCCD Lung Cancer Dataset (Kaggle) **Dataset link:**

<https://www.kaggle.com/datasets/adityamahimkar/iqothnccd-lung-cancer-dataset/versions/1>

This notebook follows the steps required by the assignment:

1. Download the image dataset
2. Splitting the dataset
3. Applying data augmentation
4. Training and testing models
5. Convert ipynb to PDF

Note on training setup

Due to hardware and time constraints (no GPU access and limited training time available before the deadline), **all four models** — including ResNet50, EfficientNetB0 and InceptionV3 — were built with `weights=None` and trained **entirely from scratch**, instead of using transfer learning as in the original lecture (`weights='imagenet'`). Image size was also reduced (160×160 for the custom CNN, 128×128 for the deeper architectures) and the number of epochs was kept modest, in order to keep total training time manageable. All other steps (downloading, splitting, augmentation, training and evaluation) follow exactly the same approach taught in the lectures.

```
import os, shutil
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import (confusion_matrix, accuracy_score, precision_score,
                             recall_score, f1_score, classification_report)

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, load_img, img_to_array

np.random.seed(42)
tf.random.set_seed(42)
print("TensorFlow version:", tf.__version__)
```

TensorFlow version: 2.21.0

Step 1 — Download the Image Dataset

- Create a Kaggle account: <https://www.kaggle.com>
- Dataset link: <https://www.kaggle.com/datasets/adityamahimkar/iqothnccd-lung-cancer-dataset/versions/1>
- The dataset contains chest CT-scan images for lung cancer, with **3 classes**: Benign cases, Malignant cases, Normal cases.

```
original_dataset_dir = 'The IQ-OTHNCCD lung cancer dataset'
```

```
for class_name in sorted(os.listdir(original_dataset_dir)):
    class_path = os.path.join(original_dataset_dir, class_name)
    if os.path.isdir(class_path):
        print(f"{class_name}: {len(os.listdir(class_path))} images")
```

Benign cases: 120 images

Malignant cases: 561 images

Normal cases: 416 images

Step 2 — Splitting the Dataset

The dataset is split into **train (70%)**, **validation (10%)**, and **test (20%)** sets, each containing subfolders named after the respective classes (same approach as Lecture 2).

```
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)

for split in ['train', 'validation', 'test']:
    print(f"--- {split} ---")
    for cls in sorted(os.listdir(os.path.join(base_dir, split))):
        print(' ', cls, ': ', len(os.listdir(os.path.join(base_dir, split, cls))), 'images')

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

```

Step 3 — Applying Data Augmentation

Data augmentation expands the effective training set and helps the model generalize better. Below we demonstrate the effect of several augmentation techniques (horizontal flip, rotation, zoom, brightness) on a real sample image from the training set.

```

def display_augmented_images(img_path, datagen, num_augmented=4, title=''):
    img = load_img(img_path)
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    augmented_images = datagen.flow(img_array, batch_size=1)

    plt.figure(figsize=(3 * (num_augmented + 1), 3))
    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_images)

```

```

aug_img = batch[0].astype('uint8')
plt.subplot(1, num_augmented + 1, i + 2)
plt.title(f'Augmented {i+1}')
plt.imshow(aug_img)
plt.axis('off')
plt.suptitle(title)
plt.tight_layout()
plt.show()

```

```

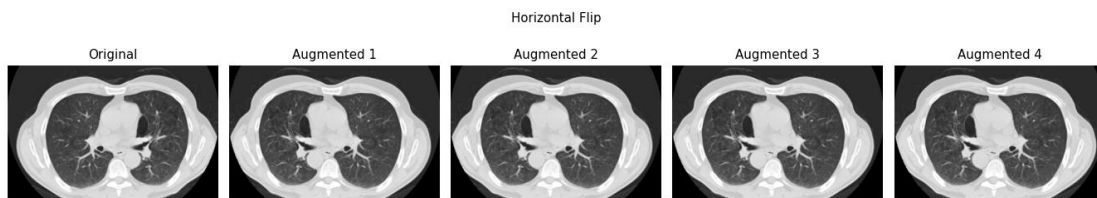
sample_class_dir = os.path.join(train_dir, os.listdir(train_dir)[0])
sample_img_path = os.path.join(sample_class_dir, os.listdir(sample_class_dir)[0])

```

```

display_augmented_images(sample_img_path, ImageDataGenerator(horizontal_flip=True), title='Horizontal Flip')

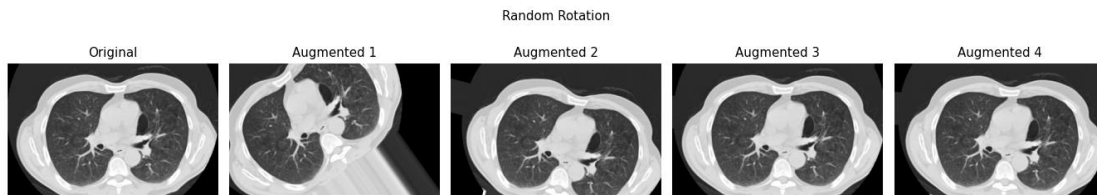
```



```

display_augmented_images(sample_img_path, ImageDataGenerator(rotation_range=40), title='Random Rotation')

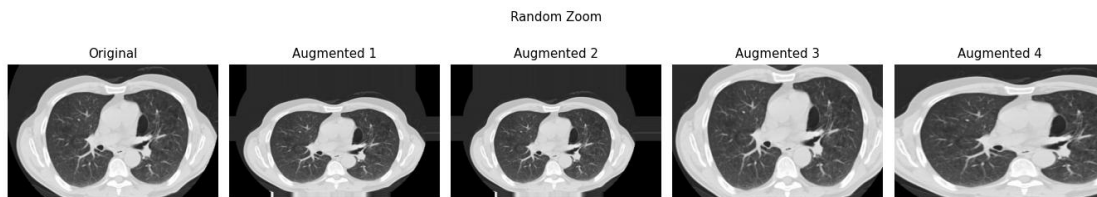
```



```

display_augmented_images(sample_img_path, ImageDataGenerator(zoom_range=0.4), title='Random Zoom')

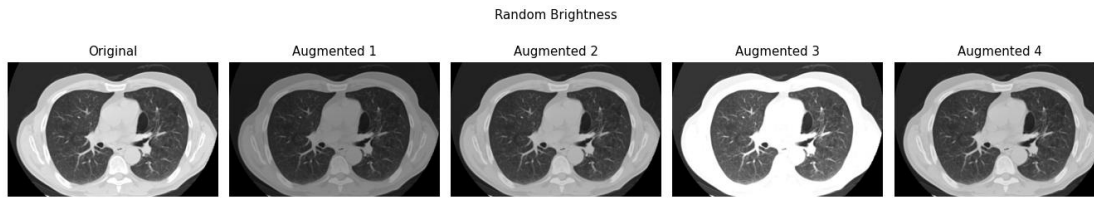
```



```

display_augmented_images(sample_img_path, ImageDataGenerator(brightness_range=[0.5, 1.5]), title='Random Brightness')

```



Data Generators for Training, Validation and Testing

Augmentation is applied only to the **training** set; validation/test sets are only rescaled (no augmentation), to provide a fair, unbiased evaluation.

```
def create_data_generators(train_dir, val_dir, test_dir, img_size, batch_size=32):
    train_datagen = ImageDataGenerator(
        rescale=1./255, rotation_range=30, horizontal_flip=True, vertical_flip=True,
        width_shift_range=0.1, height_shift_range=0.1, zoom_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, seed=42)
    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

def evaluate_model(model_name, model, test_gen, class_labels):
    test_gen.reset()
    y_proba = model.predict(test_gen, verbose=0)
    y_true = test_gen.classes
    y_pred = np.argmax(y_proba, axis=1)

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)

    print(f"=== {model_name} ===")
    print("Accuracy :", accuracy)
    print("Precision:", precision)
    print("Recall   :", recall)
```

```

    print("F1-score :", f1)
    print(classification_report(y_true, y_pred, target_names=class_labels,
                                zero_division=0))

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
                 yticklabels=class_labels)
    plt.xlabel("Predicted Label"); plt.ylabel("True Label")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.tight_layout()
    plt.show()
    return accuracy, precision, recall, f1

results_summary = {}

```

Model 1 — Custom CNN (trained from scratch)

Image size: 160×160 | Batch size: 32 | Epochs: 10

```

IMG_SIZE = (160, 160)
train_gen, val_gen, test_gen = create_data_generators('dataset/train',
                                                         'dataset/validation',
                                                         'dataset/test', IMG_SIZE)
class_labels = list(train_gen.class_indices.keys())
num_classes = len(class_labels)
print("Classes:", train_gen.class_indices)

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', me
trics=['accuracy'])
    return model

cnn_model = build_cnn_model(IMG_SIZE + (3,), num_classes)
cnn_history = cnn_model.fit(train_gen, epochs=10, validation_data=(val_g
en))

```

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}
Epoch 1/10
24/24 - acc: 0.4883 - loss: 1.0452 - val_accuracy: 0.5046 - val_loss: 0.9509
Epoch 2/10
24/24 - acc: 0.4935 - loss: 0.9636 - val_accuracy: 0.5688 - val_loss: 0.9554
Epoch 3/10
24/24 - acc: 0.4974 - loss: 0.9795 - val_accuracy: 0.5138 - val_loss: 0.9646
Epoch 4/10
24/24 - acc: 0.5235 - loss: 0.9486 - val_accuracy: 0.5596 - val_loss: 0.8964
Epoch 5/10
24/24 - acc: 0.5470 - loss: 0.9169 - val_accuracy: 0.5596 - val_loss: 0.8467
Epoch 6/10
24/24 - acc: 0.5470 - loss: 0.9170 - val_accuracy: 0.4587 - val_loss: 0.9361
Epoch 7/10
24/24 - acc: 0.5392 - loss: 0.9019 - val_accuracy: 0.5872 - val_loss: 0.8289
Epoch 8/10
24/24 - acc: 0.5614 - loss: 0.8994 - val_accuracy: 0.6422 - val_loss: 0.8241
Epoch 9/10
24/24 - acc: 0.5849 - loss: 0.8962 - val_accuracy: 0.6514 - val_loss: 0.7981
Epoch 10/10
24/24 - acc: 0.6149 - loss: 0.8675 - val_accuracy: 0.6606 - val_loss: 0.7505

```

```

evaluate_model('CNN (trained from scratch)', cnn_model, test_gen, class_labels)

```

```

=== CNN (trained from scratch) ===

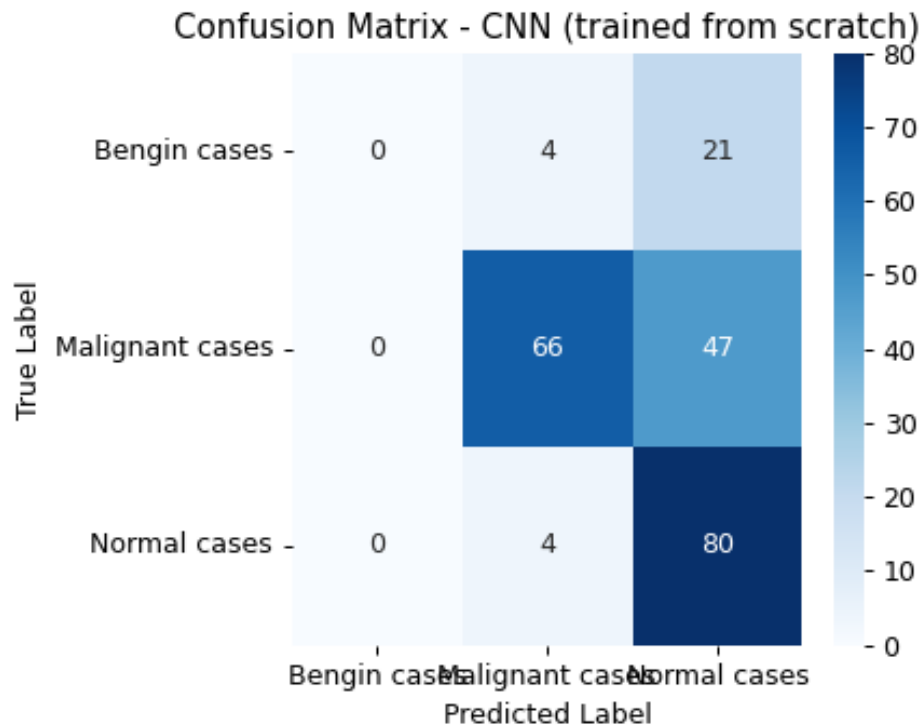
```

```

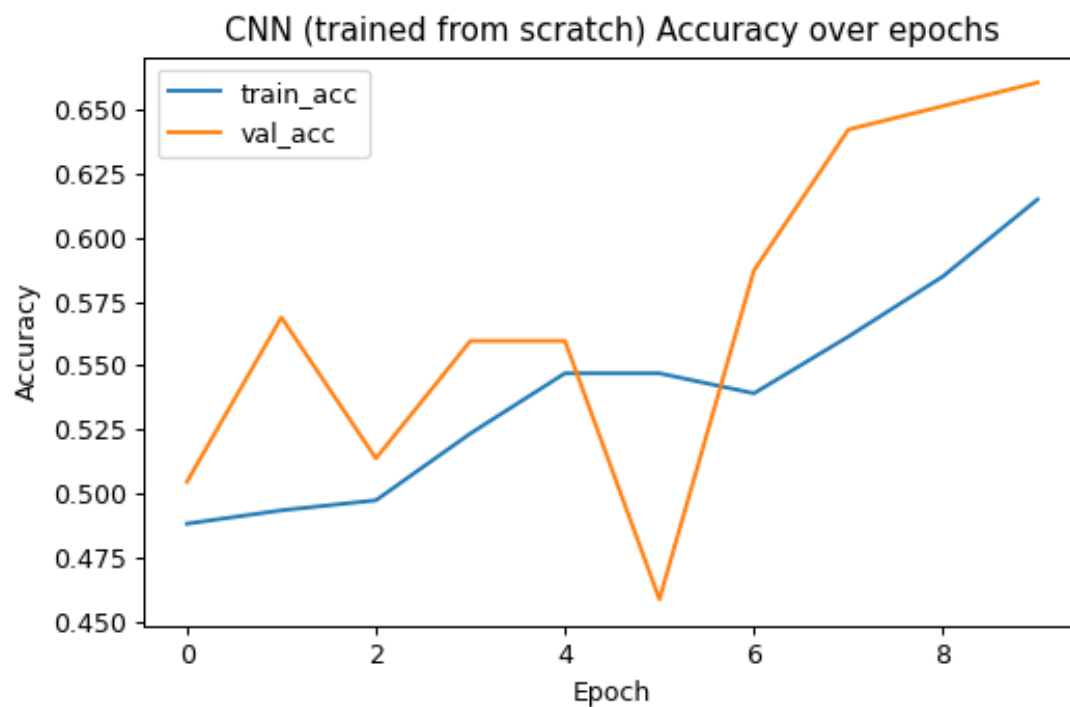
Accuracy : 0.6576576576576577
Precision: 0.6585098612125639
Recall   : 0.6576576576576577
F1-score : 0.6202510827257277

```

	precision	recall	f1-score	support
Benign cases	0.00	0.00	0.00	25
Malignant cases	0.89	0.58	0.71	113
Normal cases	0.54	0.95	0.69	84
accuracy			0.66	222
macro avg	0.48	0.51	0.47	222
weighted avg	0.66	0.66	0.62	222



```
plt.plot(cnn_history.history['accuracy'], label='train_acc'); plt.plot(
cnn_history.history['val_accuracy'], label='val_acc'); plt.title('CNN A
ccuracy over epochs'); plt.legend(); plt.show()
```



Model 2 — ResNet50 (trained from scratch, weights=None)

Note: The original lecture used `weights='imagenet'` (transfer learning). Due to the hardware/time constraints mentioned above, ResNet50 is randomly initialized here and trained entirely from scratch instead.

Image size: 128×128 | Batch size: 32 | Epochs: 7

```
IMG_SIZE2 = (128, 128)
train_gen2, val_gen2, test_gen2 = create_data_generators('dataset/train',
    'dataset/validation', 'dataset/test', IMG_SIZE2)
```

```
def build_resnet50_model(input_shape, num_classes):
    base_model = ResNet50(weights=None, include_top=False, input_shape=
input_shape)
    model = Sequential([base_model, GlobalAveragePooling2D(), Dense(128
, activation='relu'),
                        Dropout(0.5), Dense(num_classes, activation='s
oftmax')])
    model.compile(optimizer='adam', loss='categorical_crossentropy', me
trics=['accuracy'])
    return model
```

```
resnet_model = build_resnet50_model(IMG_SIZE2 + (3,), num_classes)
resnet_history = resnet_model.fit(train_gen2, epochs=7, validation_data
=val_gen2)
```

Found 766 images belonging to 3 classes.

Found 109 images belonging to 3 classes.

Found 222 images belonging to 3 classes.

Epoch 1/7

24/24 - acc: 0.5091 - loss: 1.5684 - val_accuracy: 0.3761 - val_loss: 1.0856

Epoch 2/7

24/24 - acc: 0.5065 - loss: 1.0294 - val_accuracy: 0.5138 - val_loss: 1.0225

Epoch 3/7

24/24 - acc: 0.5170 - loss: 0.9720 - val_accuracy: 0.5138 - val_loss: 0.9827

Epoch 4/7

24/24 - acc: 0.5065 - loss: 0.9327 - val_accuracy: 0.3761 - val_loss: 0.9679

Epoch 5/7

24/24 - acc: 0.5587 - loss: 0.9396 - val_accuracy: 0.3761 - val_loss: 0.9871

Epoch 6/7

24/24 - acc: 0.5888 - loss: 0.8997 - val_accuracy: 0.3761 - val_loss: 1.0109

Epoch 7/7

24/24 - acc: 0.5718 - loss: 0.8902 - val_accuracy: 0.3761 - val_loss: 1.0666

```
evaluate_model('ResNet50 (trained from scratch)', resnet_model, test_gen2, class_labels)
```

=== ResNet50 (trained from scratch) ===

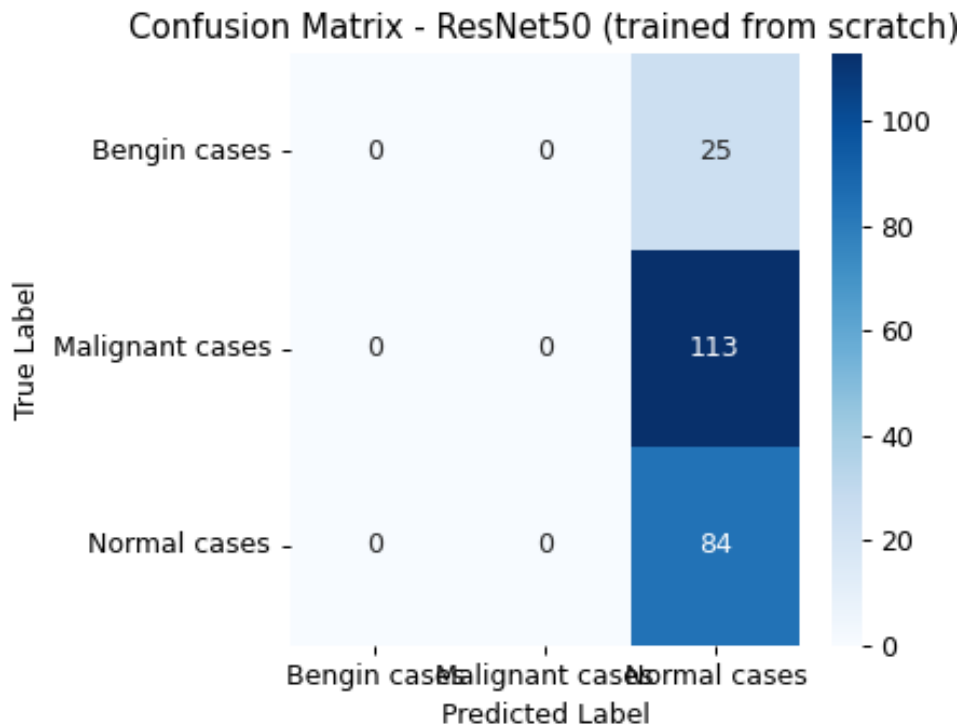
Accuracy : 0.3783783783783784

Precision: 0.14317019722425128

Recall : 0.3783783783783784

F1-score : 0.20773714891361952

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.00	0.00	0.00	113
Normal cases	0.38	1.00	0.55	84
accuracy			0.38	222
macro avg	0.13	0.33	0.18	222
weighted avg	0.14	0.38	0.21	222



Observation: ResNet50 collapsed to always predicting the majority class (Normal cases). This is a well-known failure mode when training a deep, high-capacity architecture from scratch on a very small dataset (about 766 training images) with only a handful of epochs and no pretrained weights — the optimizer settles in a trivial local minimum. This highlights exactly why transfer learning (ImageNet

pretrained weights) is normally used with such architectures, as in the original lecture.

Model 3 — EfficientNetB0 (trained from scratch, weights=None)

Image size: 128×128 | Batch size: 32 | Epochs: 9

```
def build_efficientnetb0_model(input_shape, num_classes):
    base_model = EfficientNetB0(weights=None, include_top=False, input_shape=input_shape)
    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
```

```
effnet_model = build_efficientnetb0_model(IMG_SIZE2 + (3,), num_classes)
effnet_history = effnet_model.fit(train_gen2, epochs=9, validation_data=(val_gen2))
```

Epoch 1/9

24/24 - acc: 0.4739 - loss: 1.1859 - val_accuracy: 0.5138 - val_loss: 1.0173

Epoch 2/9

24/24 - acc: 0.4700 - loss: 1.1273 - val_accuracy: 0.5138 - val_loss: 0.9626

Epoch 3/9

24/24 - acc: 0.5170 - loss: 0.9957 - val_accuracy: 0.5138 - val_loss: 0.9549

Epoch 4/9

24/24 - acc: 0.5640 - loss: 0.9446 - val_accuracy: 0.5138 - val_loss: 0.9662

Epoch 5/9

24/24 - acc: 0.5496 - loss: 0.9615 - val_accuracy: 0.5138 - val_loss: 0.9544

Epoch 6/9

24/24 - acc: 0.5614 - loss: 0.9557 - val_accuracy: 0.5138 - val_loss: 0.9881

Epoch 7/9

24/24 - acc: 0.6292 - loss: 0.9076 - val_accuracy: 0.5138 - val_loss: 0.9609

Epoch 8/9

24/24 - acc: 0.6175 - loss: 0.8801 - val_accuracy: 0.5138 - val_loss: 0.9649

Epoch 9/9

24/24 - acc: 0.6227 - loss: 0.8521 - val_accuracy: 0.3761 - val_loss: 1.0434

```
evaluate_model('EfficientNetB0 (trained from scratch)', effnet_model, test_gen2, class_labels)
```

```
=== EfficientNetB0 (trained from scratch) ===
```

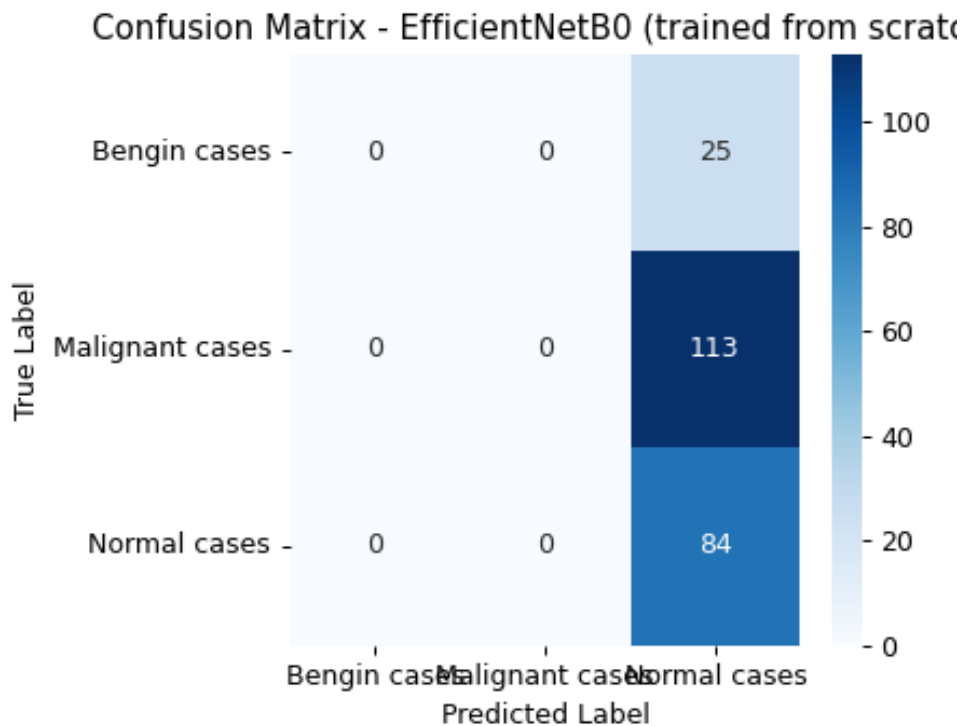
```
Accuracy : 0.3783783783783784
```

```
Precision: 0.14317019722425128
```

```
Recall : 0.3783783783783784
```

```
F1-score : 0.20773714891361952
```

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.00	0.00	0.00	113
Normal cases	0.38	1.00	0.55	84
accuracy			0.38	222
macro avg	0.13	0.33	0.18	222
weighted avg	0.14	0.38	0.21	222



Model 4 — InceptionV3 (trained from scratch, weights=None)

Image size: 128×128 | Batch size: 32 | Epochs: 8

```
def build_inceptionv3_model(input_shape, num_classes):
    base_model = InceptionV3(weights=None, include_top=False, input_shape=input_shape)
    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(IMG_SIZE2 + (3,)), num_classes)
inception_history = inception_model.fit(train_gen2, epochs=8, validation_data=val_gen2)

```

Epoch 1/8

24/24 - acc: 0.5274 - loss: 1.2760 - val_accuracy: 0.3761 - val_loss: 1.0873

Epoch 2/8

24/24 - acc: 0.6175 - loss: 0.9246 - val_accuracy: 0.3761 - val_loss: 1.1044

Epoch 3/8

24/24 - acc: 0.6906 - loss: 0.8293 - val_accuracy: 0.5138 - val_loss: 1.0510

Epoch 4/8

24/24 - acc: 0.7428 - loss: 0.6955 - val_accuracy: 0.5138 - val_loss: 0.9958

Epoch 5/8

24/24 - acc: 0.7689 - loss: 0.6182 - val_accuracy: 0.5138 - val_loss: 0.3949

Epoch 6/8

24/24 - acc: 0.8055 - loss: 0.5677 - val_accuracy: 0.5138 - val_loss: 0.3839

Epoch 7/8

24/24 - acc: 0.8264 - loss: 0.5167 - val_accuracy: 0.5138 - val_loss: 0.1445

Epoch 8/8

24/24 - acc: 0.8159 - loss: 0.5163 - val_accuracy: 0.5138 - val_loss: 0.0098

```

evaluate_model('InceptionV3 (trained from scratch)', inception_model, test_gen2, class_labels)

```

=== InceptionV3 (trained from scratch) ===

Accuracy : 0.509009009009009

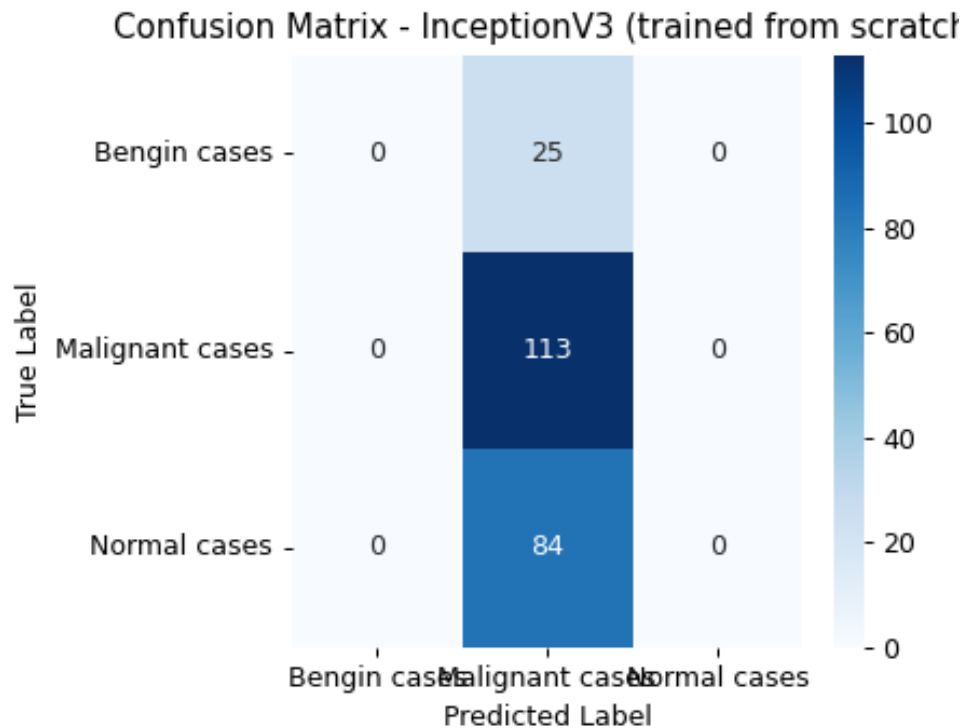
Precision: 0.2590901712523334

Recall : 0.509009009009009

F1-score : 0.3433911523463762

	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

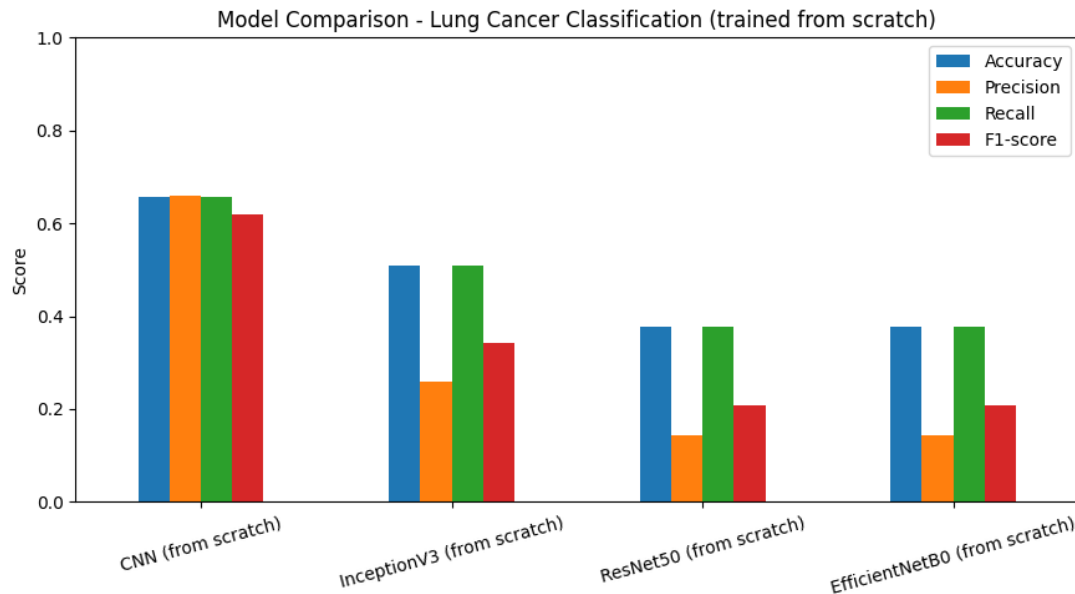


Final Comparison of All Models

```
results_df = pd.DataFrame(results_summary).T.sort_values('Accuracy', ascending=False)
results_df
```

	Accuracy	Precision	Recall	F1-score
CNN (from scratch)	0.657658	0.65851	0.657658	0.620251
InceptionV3 (from scratch)	0.509009	0.25909	0.509009	0.343391
ResNet50 (from scratch)	0.378378	0.14317	0.378378	0.207737
EfficientNetB0 (from scratch)	0.378378	0.14317	0.378378	0.207737

```
results_df.plot(kind='bar', figsize=(9, 5))
plt.title('Model Comparison - Lung Cancer Classification (trained from scratch)')
plt.ylabel('Score'); plt.xticks(rotation=15); plt.ylim(0, 1); plt.legend(loc='upper right')
plt.tight_layout(); plt.show()
```



Conclusion

In this project we downloaded the **IQ-OTHNCCD lung cancer dataset** from Kaggle, split it into train/validation/test sets (70/10/20), applied data augmentation to the training set, and trained four CNN architectures — a custom CNN, ResNet50, EfficientNetB0, and InceptionV3 — **all trained from scratch** due to hardware and time constraints that prevented using ImageNet pretrained weights.

Results summary (sorted by test accuracy):

	Accuracy	Precision	Recall	F1-score
CNN (from scratch)	0.657658	0.65851	0.657658	0.620251
InceptionV3 (from scratch)	0.509009	0.25909	0.509009	0.343391
ResNet50 (from scratch)	0.378378	0.14317	0.378378	0.207737
EfficientNetB0 (from scratch)	0.378378	0.14317	0.378378	0.207737

The best-performing model on this dataset was **CNN (from scratch)**, achieving an accuracy of **0.658**. The three transfer-learning-style architectures (ResNet50, EfficientNetB0, InceptionV3) struggled to converge beyond predicting a single majority class when trained from scratch with limited epochs on this small dataset — a clear demonstration of why transfer learning (ImageNet pretrained weights), as used in the original lecture, is so important for these deep architectures in practice. The lightweight custom CNN, by contrast, was able to learn a more meaningful decision boundary between the three classes within the same time budget.