

```

In [1]: #import laibraries
import os
import shutil

#divided data to Train / Validation / Test
from sklearn.model_selection import train_test_split

#libraries for image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import (
    confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, classification_report
)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Conv2D, MaxPooling2D, Flatten, Dense, Dropout
)
from tensorflow.keras.preprocessing.image import ImageDataGenerator

print("Libraries imported.")

```

Libraries imported.

```

In [19]: # Cell 2: Define dataset paths and create base folders

import os # import the 'os' module

# Relative path to the dataset folder (must be in the same directory as the notebook)
# Change this to an absolute path if needed.
original_dataset_dir = 'The IQ-OTHNCCD lung cancer dataset'

# ---- Sanity check: verify that the dataset folder exists ----
if not os.path.isdir(original_dataset_dir):
    print("ERROR: original_dataset_dir does NOT exist.")
    print("Current working directory:", os.getcwd())
    print("\nItems in the current folder:")
    for item in os.listdir('.'):
        print(" -", item)
    # Stop execution so you can fix the path
    raise FileNotFoundError(f"Dataset folder not found: {original_dataset_dir}")

# Base folder where the split dataset (train/validation/test) will be stored
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')

# Create base, train, validation, and test folders if they do not exist
for d in [base_dir, train_dir, validation_dir, test_dir]:
    os.makedirs(d, exist_ok=True)

# Print the path and contents of the original dataset folder to verify

```

```
print("original_dataset_dir =", os.path.abspath(original_dataset_dir))
print("Contents of original_dataset_dir:", os.listdir(original_dataset_dir))
```

```
original_dataset_dir = C:\Users\Raed.tattan\Documents\lung_Project\The IQ-OTHNCCD lung cancer dataset
Contents of original_dataset_dir: ['Benign cases', 'IQ-OTH_NCCD lung cancer dataset.txt', 'Malignant cases', 'Normal cases']
```

```
In [20]: # Cell 3: Define split function and apply it to each class

# Split ratios: 70% train, 10% validation, 20% test
train_ratio = 0.7
validation_ratio = 0.1
test_ratio = 0.2

def split_and_copy_files(class_dir, dest_base_dir):
    """
    This function:
    1. Reads all image file paths inside a single class folder (e.g., Benign).
    2. Splits the images into train / validation / test according to the given ratios.
    3. Copies the images to new folders under: dataset/train, dataset/validation, data
    """

    # Collect all image file paths from this class directory
    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))
    ]

    print(f"\nClass {os.path.basename(class_dir)} has {len(all_images)} images before

    # If there are no images, skip this class
    if len(all_images) == 0:
        print("No images found in this class. Skipping.")
        return

    # 1) Split into: Train and (Validation + Test)
    train_images, temp_images = train_test_split(
        all_images,
        test_size=(1 - train_ratio),
        random_state=42
    )

    # 2) Split (Validation + Test) into: Validation and Test
    validation_images, test_images = train_test_split(
        temp_images,
        test_size=test_ratio / (validation_ratio + test_ratio),
        random_state=42
    )

    # 3) Copy files into new folders: train / validation / test
    for images, folder_name in zip(
        [train_images, validation_images, test_images],
        ['train', 'validation', 'test']
    ):
        # Example: dataset/train/Benign
        dest_dir = os.path.join(
            dest_base_dir,
            folder_name,
```

```

        os.path.basename(class_dir)
    )
    os.makedirs(dest_dir, exist_ok=True)

    for file_path in images:
        shutil.copy(file_path, dest_dir)

    print(f"Copied images -> train={len(train_images)}, val={len(validation_images)},

print("\nStarting class processing...")

# Apply the split function to each class folder inside the original dataset directory
for class_name in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_name)
    if os.path.isdir(full_class_dir):
        print("Processing class:", class_name)
        split_and_copy_files(full_class_dir, base_dir)

print("\nDone splitting dataset.")

```

Starting class processing...
Processing class: Bengin cases

Class Bengin cases has 120 images before splitting.
Copied images -> train=83, val=12, test=25
Processing class: Malignant cases

Class Malignant cases has 561 images before splitting.
Copied images -> train=392, val=56, test=113
Processing class: Normal cases

Class Normal cases has 416 images before splitting.
Copied images -> train=291, val=41, test=84

Done splitting dataset.

In [21]: *# Cell 4: Print the number of images in each split (train / validation / test)*

```

print("Image counts after splitting:")

for split in ['train', 'validation', 'test']:
    split_dir = os.path.join(base_dir, split)
    total = 0
    print(f"\nSplit: {split}")

    for class_name in os.listdir(split_dir):
        cdir = os.path.join(split_dir, class_name)
        if os.path.isdir(cdir):
            count = len(os.listdir(cdir))
            total += count
            print(f"  Class {class_name}: {count}")

    print("  Total images in", split, "=", total)

```

Image counts after splitting:

Split: train

Class Benign cases: 83
Class Malignant cases: 392
Class Normal cases: 291
Total images in train = 766

Split: validation

Class Benign cases: 12
Class Malignant cases: 56
Class Normal cases: 41
Total images in validation = 109

Split: test

Class Benign cases: 25
Class Malignant cases: 113
Class Normal cases: 84
Total images in test = 222

```
In [22]: # Cell 5: Create data generators with data augmentation

# Target image size (height, width)
img_size = (240, 240)
batch_size = 32

# Data augmentation for training set
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    zoom_range=0.2
)

# Only rescaling for validation and test sets (no augmentation)
val_test_datagen = ImageDataGenerator(rescale=1./255)

# Training generator (will apply augmentation)
train_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True
)

# Validation generator
val_gen = val_test_datagen.flow_from_directory(
    validation_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Test generator
test_gen = val_test_datagen.flow_from_directory(
```

```

    test_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

print("Class indices:", train_gen.class_indices)

num_classes = len(train_gen.class_indices)
input_shape = (img_size[0], img_size[1], 3)

```

Found 766 images belonging to 3 classes.
Found 109 images belonging to 3 classes.
Found 222 images belonging to 3 classes.
Class indices: {'Benign cases': 0, 'Malignant cases': 1, 'Normal cases': 2}

In [23]: *# Cell 6: Build a simple CNN model*

```

def build_cnn_model(input_shape, num_classes):
    """
    Build a simple CNN classifier for the lung cancer dataset.
    """
    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

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

```

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)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 58, 58, 64)	0
conv2d_2 (Conv2D)	(None, 56, 56, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 128)	0
flatten (Flatten)	(None, 100352)	0
dense (Dense)	(None, 128)	12845184
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 3)	387

=====
Total params: 12938819 (49.36 MB)
Trainable params: 12938819 (49.36 MB)
Non-trainable params: 0 (0.00 Byte)

```
In [24]: # Cell 7: Train the CNN model

epochs = 10 # You can reduce this (e.g., 5) if training is too slow

history_cnn = cnn_model.fit(
    train_gen,
    epochs=epochs,
    validation_data=val_gen
)
```

```

Epoch 1/10
24/24 [=====] - 51s 2s/step - loss: 1.3207 - accuracy: 0.489
6 - val_loss: 0.9511 - val_accuracy: 0.5138
Epoch 2/10
24/24 [=====] - 47s 2s/step - loss: 0.9725 - accuracy: 0.492
2 - val_loss: 0.9318 - val_accuracy: 0.5138
Epoch 3/10
24/24 [=====] - 47s 2s/step - loss: 0.9556 - accuracy: 0.518
3 - val_loss: 0.9101 - val_accuracy: 0.5046
Epoch 4/10
24/24 [=====] - 45s 2s/step - loss: 0.9340 - accuracy: 0.502
6 - val_loss: 0.8796 - val_accuracy: 0.4862
Epoch 5/10
24/24 [=====] - 44s 2s/step - loss: 0.9222 - accuracy: 0.524
8 - val_loss: 0.8635 - val_accuracy: 0.5963
Epoch 6/10
24/24 [=====] - 44s 2s/step - loss: 0.9208 - accuracy: 0.556
1 - val_loss: 0.9608 - val_accuracy: 0.5321
Epoch 7/10
24/24 [=====] - 45s 2s/step - loss: 0.9109 - accuracy: 0.540
5 - val_loss: 0.8919 - val_accuracy: 0.5505
Epoch 8/10
24/24 [=====] - 44s 2s/step - loss: 0.9604 - accuracy: 0.573
1 - val_loss: 0.9787 - val_accuracy: 0.4954
Epoch 9/10
24/24 [=====] - 47s 2s/step - loss: 0.9157 - accuracy: 0.561
4 - val_loss: 0.8453 - val_accuracy: 0.5321
Epoch 10/10
24/24 [=====] - 45s 2s/step - loss: 0.8863 - accuracy: 0.553
5 - val_loss: 0.8239 - val_accuracy: 0.5780

```

```

In [25]: # Cell 8: Evaluate the model on the test set and print metrics

# Get predicted probabilities for test set
y_pred_probs = cnn_model.predict(test_gen)

# Convert probabilities to predicted class indices
y_pred_classes = np.argmax(y_pred_probs, axis=1)

# True class indices from the generator
y_true = test_gen.classes

# Class labels (names)
labels = list(test_gen.class_indices.keys())

# Calculate metrics
accuracy = accuracy_score(y_true, y_pred_classes)
precision = precision_score(y_true, y_pred_classes, average='weighted')
recall = recall_score(y_true, y_pred_classes, average='weighted')
f1 = f1_score(y_true, y_pred_classes, average='weighted')
cm = confusion_matrix(y_true, y_pred_classes)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

print("\nClassification 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")
plt.ylabel("True")
plt.title("Confusion Matrix - CNN")
plt.show()
```

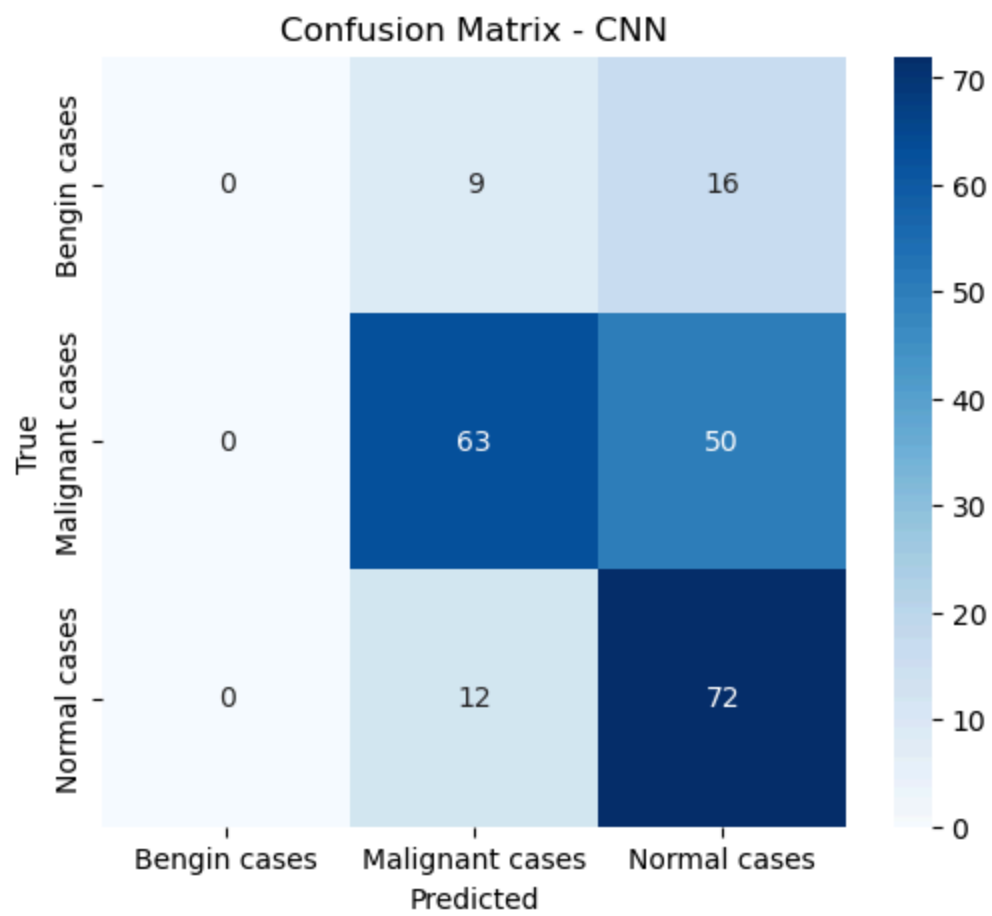
7/7 [=====] - 3s 350ms/step

Accuracy: 0.6081081081081081
Precision: 0.579171562867215
Recall: 0.6081081081081081
F1-score: 0.5709936854126729

Classification Report:

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.75	0.56	0.64	113
Normal cases	0.52	0.86	0.65	84
accuracy			0.61	222
macro avg	0.42	0.47	0.43	222
weighted avg	0.58	0.61	0.57	222

```
C:\Users\Raed.tattan\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Raed.tattan\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Raed.tattan\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Raed.tattan\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.
    _warn_prf(average, modifier, msg_start, len(result))
```



In []: