RIPHAH INTERNATIONAL UNIVERSITY

ARTIFICIAL INTERNATIONAL

LUNG CANCER CLASSIFICATION SYSTEM

DOCUMENTATION

SUBMITTED TO

SIR JUNAID

SUBMITTED BY

MUHAMMAD ABDULLAH NADEEM (66156)

USMAN AKRAM

(65587)

1. Workflow Overview

The solution follows a standard Deep Learning pipeline for image classification:

1.1. Ingestion:

The script uses kagglehub to download the "chest-ctscan-images" dataset. It then automatically traverses the directory structure to identify the correct paths for the training and validation/test splits, handling variations in folder naming conventions (e.g., 'valid' vs 'test').

1.2. Preprocessing:

An ImageDataGenerator pipeline is established. For training data, it applies efficientnet.preprocess_input for normalization and performs real-time data augmentation (rotation, zoom, shear, horizontal flip) to increase dataset diversity and model robustness. Validation data is only normalized.

1.3. Modelling:

The architecture is based on **EfficientNetB0**.

- **Backbone:** The pre-trained EfficientNetB0 (on ImageNet) is instantiated without its top classification layer.
- **Fine-Tuning:** The initial layers are frozen to retain learned features, while the top 20 layers are unfrozen to fine-tune the model on the specific textures of CT scans.
- **Custom Head:** A new head is added, comprising a GlobalAveragePooling2D layer, a Dropout layer (rate 0.5) for regularization, and a final Dense softmax layer for 4-class classification.

1.4. Training:

The model is compiled with the Adam optimizer (learning rate 1e-4) and Categorical Crossentropy loss. The training loop utilizes callbacks: EarlyStopping to halt training if validation loss stagnates, and ReduceLROnPlateau to lower the learning rate when a plateau is detected.

1.5. Evaluation:

Post-training, the model is evaluated on the validation set. Visualizations include accuracy/loss curves over epochs and a Confusion Matrix heatmap. A text-based classification report details Precision, Recall, and F1-score per class.

2. Key Functions

Function Name	**Description**
download_and_locate_data	Handles dataset download via kagglehub and dynamically resolves the paths to the 'train' and 'valid' (or 'test') directories.
create_generators	Configures ImageDataGenerator instances for training (with augmentation) and validation (preprocessing only). Returns data iterators (train_gen, val_gen).
build_model	Assembles the CNN. It loads the EfficientNetB0 base, configures layer freezing/unfreezing for fine-tuning, and appends the custom classification layers.

Function Name	Description
train_model	Executes the training process using model.fit. It sets up the EarlyStopping and ReduceLROnPlateau callbacks.
evaluate_model	Runs inference on the validation generator. It computes predictions, generates the confusion matrix using sklearn.metrics, and prints the classification report.

3. Inputs and Outputs

Inputs:

- A dataset directory containing subfolders for each class ('adenocarcinoma', 'large.cell.carcinoma', 'squamous.cell.carcinoma', 'normal').
- **Format:** Images (RGB), automatically resized to **224x224 pixels** during loading.

Outputs:

- **Trained Model:** A TensorFlow/Keras model object with optimized weights.
- **Visualizations:**

Graphs displaying Training vs. Validation Accuracy and Loss.

A Heatmap of the Confusion Matrix comparing true vs. predicted labels.

- **Metrics:** A printed report showing Precision, Recall, and F1-Score for all 4 classes.

4. Dependencies

To run this code, the following libraries are required:

- tensorflow: The core framework for building and training the neural network.
- kagglehub: To download the specific dataset from Kaggle.
- numpy: For numerical operations and array handling.
- matplotlib & seaborn: For plotting graphs and the confusion matrix.
- scikit-learn: For calculating evaluation metrics (classification report, confusion matrix).

5. Execution Steps

5.1. Environment Setup

Ensure Python and necessary libraries (tensorflow, kagglehub, matplotlib, seaborn, scikit-learn) are installed. A GPU-enabled environment (like Google Colab) is recommended for faster training.

5.2. Run Script:

Execute the Python script.

- It will download the dataset.
- It will preprocess the data and build the model.
- Training will proceed for up to 20 epochs (or fewer if early stopping triggers).

5.3. Review Results:

After training, the script will display the training curves, print the classification metrics, and show the confusion matrix heatmap.