1. Title

Coffee Plant

1. Description

Dataset and Availability

We utilized the BRACOL dataset, a publicly available dataset of coffee leaf images. The dataset can be accessed at the following DOI: [https://data.mendeley.com/datasets/c5yvn32dzg/2].

1. Dataset Information.

We utilized the BRACOL dataset, a publicly available dataset of coffee leaf images. The dataset can be accessed at the following DOI: [https://data.mendeley.com/datasets/c5yvn32dzg/2].

1. Code Information

Programming Language: Matlab 2023b

### Usage Instructions

### In Matlab open file and execute

### Requirements

### Requirements – need install machine learning library from the internet (from [MathWorks](https://au.mathworks.com/products/matlab.html) site)

### Methodology

### Return to the paper

### Code

### Main\_m

This MATLAB code implements a **transfer learning pipeline using AlexNet** for image classification. It reads a dataset, modifies the pre-trained AlexNet to suit a new classification task, trains the model, evaluates its performance, and plots results like ROC and confusion matrix. For another models just change the **AlexNet model with others and add suitable parameters.**

### **1. Initialization**

clc

clear

close all

* Clears the command window, workspace, and closes all figures.

### **2. Load Dataset**

imds = imageDatastore('C:\...\CoffeeClass', 'IncludeSubfolders', true,'LabelSource', 'foldernames');

* Loads the image dataset from the specified folder.
* Images are automatically labeled based on their **subfolder names**.

### **3. Preprocessing Function**

imds.ReadFcn = @(filename)preprocess\_DB(filename);

* Sets a custom preprocessing function (preprocess\_DB) for all images.

### **4. Split Dataset**

[trainingImages, testImages] = splitEachLabel(imds, 0.7, 'randomize');

* Splits the dataset into **70% training** and **30% testing** sets.

### **5. Load Pre-trained AlexNet**

net = alexnet;

inputSize = net.Layers(1).InputSize

layers = net.Layers

* Loads the AlexNet model.
* Gets input size and layers for reference.

### **6. Convert to Layer Graph & Modify**

if isa(net,'SeriesNetwork')

lgraph = layerGraph(net.Layers);

else

lgraph = layerGraph(net);

end

* Converts the network to a layerGraph (required for layer editing).

### **7. Remove Final Layers**

lgraph = removeLayers(lgraph, {'fc8','prob','output'});

* Removes AlexNet's original classification layers (designed for 1000 classes).

### **8. Add New Layers**

newLayers = [

fullyConnectedLayer(6,'Name','fc','WeightLearnRateFactor',10,'BiasLearnRateFactor', 10)

softmaxLayer('Name','softmax')

classificationLayer('Name','classoutput')];

* Adds a new fully connected layer with **6 output classes**, followed by softmax and classification layers.

### **9. Connect Layers**

lgraph = addLayers(lgraph,newLayers);

lgraph = connectLayers(lgraph,'drop7','fc');

* Adds the new layers to the graph and connects them to the previous layer (drop7).

**10. Train the Network**

Coeffe\_alexnet = trainNetworks(trainingImages,lgraph, opts);

* Trains the modified AlexNet on the training dataset using specified training options (opts, not shown but assumed defined earlier).

### **11. Evaluate Accuracy**

predictedLabels = classify(Coeffe\_alexnet, testImage);

accuracy\_alexnet = mean(predictedLabels == testImages.Labels)\*100

### **12. ROC Curve and AUC**

x = double(testImages.Labels);

y = double(predictedLabels);

[FP\_rate, TP\_rate, ~, AUC] = perfcurve(x, y, 6);

plot(FP\_rate, TP\_rate, 'b-');

### **13. Confusion Matrix**

cm\_LSTM = confusionchart(testImages.Labels, predictedLabels);

cm\_LSTM.Title = 'Confusion Matrix of Renet18 Network';

### main2

load lab

* Loads a file named lab.mat that likely contains the class labels for the data.
* These labels are typically used for training or evaluating classifiers.

load features\_all\_efficientnetb0

* Loads a .mat file containing feature vectors extracted using the EfficientNetB0 model.
* This data is stored in a variable named features\_all\_efficientnetb0.

load features\_all\_googlenet

* Loads a .mat file containing feature vectors extracted using GoogLeNet.
* The variable features\_all\_googlenet holds this data.

all\_features=[features\_all\_efficientnetb0 features\_all\_googlenet];

* Concatenates the two sets of features horizontally (i.e., column-wise).
* The result is a combined feature vector for each sample, merging the information learned by both models.

save('all\_features.mat','all\_features','-v7.3')

* Saves the combined feature matrix to a file called all\_features.mat.
* The -v7.3 flag ensures compatibility for large variables (>2GB), enabling storage of big datasets.

### Main1

### Like main\_m but different model

### Main1\_g

This MATLAB script performs binary image classification using transfer learning with EfficientNetB0 on a coffee dataset. It includes steps to load the dataset, modify the network architecture, train it, evaluate it, and visualize results. Here's a complete breakdown:

1. Initialization

clc

clear

close all

* Clears the command window, variables, and closes all figures to reset the environment.

2. Load Image Dataset

imds = imageDatastore('C:\...\Coffee DB Binary', 'IncludeSubfolders', true,'LabelSource', 'foldernames');

* Loads images from the specified directory.
* Automatically assigns labels based on folder names.
* Suitable for binary classification since folders contain two classes.

3. Preprocessing Function

imds.ReadFcn = @(filename)preprocess\_DB(filename);

* Applies a custom preprocessing function preprocess\_DB, likely used for:
  + Resizing to match input size.
  + Normalization or data augmentation.

4. Split the Dataset

[trainingImages, testImages] = splitEachLabel(imds, 0.7, 'randomize');

* 70% of data used for training, 30% for testing.
* Split is randomized to ensure balanced class distribution.

5. Load Pretrained EfficientNetB0

net = efficientnetb0;

inputSize = net.Layers(1).InputSize

layers = net.Layers

* Loads the EfficientNetB0 network pretrained on ImageNet.
* Displays input image size and layers.

6. Convert to Layer Graph

if isa(net,'SeriesNetwork')

lgraph = layerGraph(net.Layers);

else

lgraph = layerGraph(net);

end

* Converts the pretrained network to a layer graph for editing.
* This is necessary to modify or replace certain layers.

7. Modify the Network

lgraph = removeLayers(lgraph, {'efficientnet-b0|model|head|dense|MatMul','Softmax','classification'});

* Removes the final classification layers meant for ImageNet’s 1000 classes:
  + The dense (fully connected) layer
  + Softmax
  + Classification layer

8. Add New Layers for Binary Classification

newLayers = [

fullyConnectedLayer(2,'Name','fc','WeightLearnRateFactor',10,'BiasLearnRateFactor', 10)

softmaxLayer('Name','softmax')

classificationLayer('Name','classoutput')];

* Adds a new fully connected layer with 2 outputs for binary classification.
* Includes softmax and a classification layer.
* Learning rates for weights and biases are increased to encourage faster adaptation of new layers.

9. Connect New Layers

lgraph = addLayers(lgraph,newLayers);

lgraph = connectLayers(lgraph,'efficientnet-b0|model|head|global\_average\_pooling2d|GlobAvgPool','fc');

* Adds new layers to the graph.
* Connects them after the global average pooling layer of EfficientNetB0.

10. Set Training Options

opts = trainingOptions('sgdm', ...

'ExecutionEnvironment','gpu', ...

'InitialLearnRate', 0.001,'MiniBatchSize',20,'MaxEpochs',10,'Plots','training-progress');

* Sets training using Stochastic Gradient Descent with Momentum (SGDM).
* Runs on GPU, with:
  + Learning rate: 0.001
  + Batch size: 20

11. Train the Network

Coeffe\_efficientnetb0 = trainNetwork(trainingImages,lgraph, opts);

save('Coeffe\_efficientnetb0.mat','Coeffe\_efficientnetb0','-v7.3');

* Trains the modified EfficientNetB0 on the training data.
* Saves the trained model to a .mat file.

12. Evaluate Model Accuracy

predictedLabels = classify(Coeffe\_efficientnetb0, testImages);

accuracy\_efficientnetb0 = mean(predictedLabels == testImages.Labels)\*100

save('accuracy\_efficientnetb0.mat','accuracy\_efficientnetb0','-v7.3')

* Predicts the class of the test images.
* Calculates the classification accuracy.
* Saves the result.

13. ROC Curve & AUC

x = double(testImages.Labels);

y = double(predictedLabels);

[FP\_rate, TP\_rate, ~, AUC] = perfcurve(x, y, 2);

plot(FP\_rate, TP\_rate);

title("ROC of efficientnetb0")

* Converts labels to numeric.
* Calculates ROC curve and AUC for class 2.
* Plots ROC and saves AUC.

save('AUC.mot','AUC','-v7.3')

14. Confusion Matrix

cm\_LSTM = confusionchart(testImages.Labels,predictedLabels);

cm\_LSTM.Title = 'Confusion Matrix of efficientnetb0 Network';

* Shows a confusion matrix to visualize classification performance.
* The title is correctly labeled for EfficientNetB0 this time (unlike the previous version with AlexNet).