Computer Vision Concepts Implementation - Assignment 1 (8890 CVIA PG)

Author: Data Man

Date: 13/05/2022

Dataset Preparation

```
% Set path for dataset folder
% path = "assets\CUB_200_2011\";
path = "..\assets\CUB_200_2011_Subset20classes\";  % for debuging

% Show DL trainging info
verbose = false;

% Read image file paths
images = readtable(fullfile(path, "images.txt"), ...
    "ReadVariableNames", false);
images.Properties.VariableNames = ["index", "file"];

% Read object bounding box info
bboxes = readtable(path + "bounding_boxes.txt", ...
    'ReadVariableNames', false);
bboxes.Properties.VariableNames = {'index', 'x', 'y', 'w', 'h'};
bm = mapBboxes(images(:, 2), bboxes(:, 2:5));

% Target size of input images
ts = [128, 128];
% ts = [256, 256];

% Datastore of full images
fids = imageDatastore(fullfile(path, "images", images.file), ...
    "labelSource", "foldernames");
fids.ReadFcn = @(file) readImages(file, ts);

% Datastore of cropped images (via bounding boxes)
cids = imageDatastore(fullfile(path, "images", images.file), ...
    "labelSource", "foldernames");
cids.ReadFcn = @(file) readImages(file, ts, bm);

% Split datasets
[trainFull, validateFull, testFull] = splitEachLabel(fids, 0.6, 0.2);
[trainCropped, validateCropped, testCropped] = splitEachLabel(cids, 0.6, 0.2);

% Split for 5-fold validation
[part1, part2, part3, part4, part5] = splitEachLabel(cids, 0.2, 0.2, 0.2, 0.2);

disp("Data preparation finish.")
```

Task 1: Classic Machine Learning Approach

```
% Read sample image
img = fids.read();

% Extract HOG features and HOG visualization
[hog_64x64, vis64x64] = extractHOGFeatures(img, 'CellSize', [64 64]);
[hog_16x16, vis16x16] = extractHOGFeatures(img, 'CellSize', [16 16]);
[hog_4x4, vis4x4] = extractHOGFeatures(img, 'CellSize', [4 4]);

% Show the original image
figure;
subplot(2,3,1:3); imshow(img);

% Visualize the HOG features
subplot(2,3,4);
plot(vis64x64);
title({'CellSize = [64 64]'; ['Length = ' num2str(length(hog_64x64))]});

subplot(2,3,5);
plot(vis16x16);
title({'CellSize = [16 16]'; ['Length = ' num2str(length(hog_16x16))]});

subplot(2,3,6);
plot(vis4x4);
title({'CellSize = [4 4]'; ['Length = ' num2str(length(hog_4x4))]});

% Set cell size of HOG features for experiments
cs = {[64 64] [16 16] [4 4]};
fs = {length(hog_64x64) length(hog_16x16) length(hog_4x4)};
```

Task 2: Deep Learning Approach

```
% Setup GPU
if (gpuDeviceCount() > 0)
    device = gpuDevice(1);
    reset(device);
end

% Network layers
layers = [
    imageInputLayer([ts 3])

    convolution2dLayer(3, 8, 'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2, 'Stride', 2)
```

```
    convolution2dLayer(3, 16, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3, 64, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3, 128, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3, 256, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3, 512, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(20)
    softmaxLayer
    classificationLayer];
```

Whole Image as Input

Experiment 1

```
disp("Experiment 1 start...")

% Conduct tests on differernt cell size
for i = 1:3
    disp("Test #" + string(i))
    cellSize = cs{i};
    featureSize = fs{i};
    disp("  Cell size: " + cellSize(1))
```

```
    disp("  Feature size: " + featureSize)
    runProcessML(trainFull, testFull, cellSize, featureSize);
end

disp("Experiment 1 finish.")
```

Experiment 2

```
disp("Experiment 2 start...")

% Combine datastores for the model
trainFullDS = combine(trainFull, arrayDatastore(trainFull.Labels));
validateFullDS = combine(validateFull, arrayDatastore(validateFull.Labels));
testFullDS = combine(testFull, arrayDatastore(testFull.Labels));

% Run deep learning training process
runProcessDL(layers, verbose, trainFullDS, validateFullDS, testFullDS);

disp("Experiment 2 finish.")
```

Target Area Image as Input

Experiment 3

```
disp("Experiment 3 start...")

% Conduct tests on differernt cell size
for i = 1:3
    disp("Test #" + string(i))
    cellSize = cs{i};
    featureSize = fs{i};
    disp("  Cell size: " + cellSize(1))
    disp("  Feature size: " + featureSize)
    runProcessML(trainCropped, testCropped, cellSize, featureSize);
end

disp("Experiment 3 finish.")
```

Experiment 4

```
disp("Experiment 4 start...")

% Combine datastores for the model
trainCroppedDS = combine(trainCropped, arrayDatastore(trainCropped.Labels));
validateCroppedDS = combine(validateCropped, arrayDatastore(validateCropped.Labels));
testCroppedDS = combine(testCropped, arrayDatastore(testCropped.Labels));

% Run deep learning training process
runProcessDL(layers, verbose, trainCroppedDS, validateCroppedDS, testCroppedDS);
```

```
disp("Experiment 4 finish.")
```

Cross-Validation Experiment

Experiment 5

```
disp("Experiment 5 start.")

% All partitions
parts = {part1, part2, part3, part4, part5};
```

Run 1: Parts 1-3 for training, Part 4 for validation, Part 5 for test

```
disp("Run #1")
run1 = runProcessDL(layers, verbose, getCVData(1:3, 4, 5, parts, ts, bm));
```

Run 2: Parts 2-4 for training, Part 5 for validation, Part 1 for test

```
disp("Run #2")
run2 = runProcessDL(layers, verbose, getCVData(2:4, 5, 1, parts, ts, bm));
```

Run 3: Parts 3-5 for training, Part 1 for validation, Part 2 for test

```
disp("Run #3")
run3 = runProcessDL(layers, verbose, getCVData(3:5, 1, 2, parts, ts, bm));
```

Run 4: Parts 4, 5, and 1 for training, Part 2 for validation, Part 3 for test

```
disp("Run #4")
run4 = runProcessDL(layers, verbose, getCVData([4 5 1], 2, 3, parts, ts, bm));
```

Run 5: Parts 5, 1, and 2 for training, Part 3 for validation, Part 4 for test

```
disp("Run #5")
run5 = runProcessDL(layers, verbose, getCVData([5 1 2], 3, 4, parts, ts, bm));
```

Cross-Validation result

```
% Cross-validation result
disp("Cross-Validatio:")
oa = mean([run1 run2 run3 run4 run5]);
disp("  Overall Accuracy (5-fold CV): "+ string(round(oa*100, 1)) +"%")

disp("Experiment 5 finish.")
```

Helper Functions

```
function map = mapBboxes(img, box)
    map = containers.Map;
    for i = 1:height(img)
```

```
        key = split(img{i, "file"}, ["\", "/"]);
        key = key{end};
        map(key) = [box{i, :}];
    end
end

% Modified read function for imageDatastore
function output = readImages(input, targetSize, varargin)
    img = imread(input);

    if nargin == 3
        file = split(input, ["\", "/"]);
        map = varargin{1};
        box = map(file{end});

        if box(1) > size(img, 2) || box(2) > size(img, 1)
            disp("error")
        end

        % Crop image by bounding boxes
        img = imcrop(img, [box(1), box(2), box(3), box(4)]);
    end

    % Duplicate grayscale images to keep data structure consistancy
    if size(img, 3) < 3
        img = cat(3, img, img, img);
    end
        output = imresize(img, targetSize);
end

% Wrap deep learning process
function restult = runProcessDL(layers, verbose, varargin)
    if nargin < 4
        training = varargin{1}{1};
        validation = varargin{1}{2};
        testing = varargin{1}{3};
    else
        training = varargin{1};
        validation = varargin{2};
        testing = varargin{3};
    end

    % Training options
    options = trainingOptions('sgdm', ...
        'InitialLearnRate', 0.001, ...
        'MiniBatchSize', 20, ...
        'MaxEpochs', 10, ...
        'Shuffle', 'every-epoch', ...
        'ValidationData', validation, ...
        'VerboseFrequency', 1, ...
```

```
        'Verbose', verbose);

    % Train the model
    model = trainNetwork(training, layers, options);

    % Display test results
    YPred = classify(model, testing);
    YTest = testing.UnderlyingDatastores{1, 1}.Labels;

    accuracy = sum(YPred == YTest)/numel(YTest);
    disp("  Overall Accuracy: "+ string(round(accuracy*100, 1)) +"%")

    restult = accuracy;
end

% Split datasets for 5-fold cross-validation
function results = getCVData( ...
    trainingPartition, validationPartition, testingPartition, parts, ts, bm)

    % Combine training set
    trds = imageDatastore(cat(1, ...
        parts{trainingPartition(1)}.Files, ...
        parts{trainingPartition(2)}.Files, ...
        parts{trainingPartition(3)}.Files));
    trds.ReadFcn = @(file) readImages(file, ts, bm);

    trlbds = arrayDatastore(cat(1, ...
        parts{trainingPartition(1)}.Labels, ...
        parts{trainingPartition(2)}.Labels, ...
        parts{trainingPartition(3)}.Labels));

    training = combine(trds, trlbds);

    % Combine validation set
    vdds = imageDatastore(parts{validationPartition}.Files);
    vdds.ReadFcn = @(file) readImages(file, ts, bm);
    vdlbds = arrayDatastore(parts{validationPartition}.Labels);
    valitation = combine(vdds, vdlbds);

    % Combine testing set
    tsds = imageDatastore(parts{testingPartition}.Files);
    tsds.ReadFcn = @(file) readImages(file, ts, bm);
    tsds.Labels = parts{testingPartition}.Labels;
    tslbds = arrayDatastore(parts{testingPartition}.Labels);
    testing = combine(tsds, tslbds);

    results = {training, valitation, testing};
end

% Wrap HOG-based classifier machine learning process
```

```
function result = runProcessML(training, testing, cellSize, featureSize)
    % Extract HOG features
    numImages = numel(training.Files);
    trainingFeatures = zeros(numImages, featureSize, 'single');

    for i = 1:numImages
        img = readimage(training, i);

        img = im2gray(img);

        % Apply pre-processing steps
        img = imbinarize(img);

        trainingFeatures(i, :) = extractHOGFeatures(img, 'CellSize', cellSize);
    end

    % Get labels for each image.
    trainingLabels = training.Labels;

    % Fit the model using SVM learner
    classifier = fitcecoc(trainingFeatures, trainingLabels);

    % Extract HOG features for test set
    [testFeatures, testLabels] = helperExtractHOGFeaturesFromImageSet( ...
        testing, featureSize, cellSize);

    % Predict on test set
    predictedLabels = predict(classifier, testFeatures);

    % Show result
    accuracy = sum(predictedLabels == testLabels)/numel(testLabels);
    disp("  Overall Accuracy: "+ string(round(accuracy*100, 1)) +"%")

    result = accuracy;
end

% Extract HOG features from an imageDatastore.
function [features, setLabels] = helperExtractHOGFeaturesFromImageSet(imds, hogFeatureSize, ce
    setLabels = imds.Labels;
    numImages = numel(imds.Files);
    features  = zeros(numImages,hogFeatureSize,'single');

    % Process each image and extract features
    for j = 1:numImages
        img = readimage(imds,j);
        img = im2gray(img);

        % Apply pre-processing steps
        img = imbinarize(img);
```

```
        features(j, :) = extractHOGFeatures(img,'CellSize',cellSize);
    end
end
```