LAMPIRAN

Source Code

SVM & PCA

```
clc;
% Define the main folder containing the images
mainFolder = 'D:\Skripsi\img_test';

% Define the categories
categories = {'matang', 'mentah', 'busuk'};

% Load the image data
imds = imageDatastore(fullfile(mainFolder, categories), 'LabelSource',
'foldernames');

% Convert images to grayscale
imds.ReadFcn = @(filename) rgb2gray(imread(filename));


[imdsTrain, imdsTest] = splitEachLabel(imds, 0.8, 'randomized');

% Extract features
net = squeezenet;
inputSize = net.Layers(1).InputSize;

% Augment data to match the input size of the network
augimdsTrain = augmentedImageDatastore([inputSize(1:2) 1], imdsTrain,
'ColorPreprocessing', 'gray2rgb');
augimdsTest = augmentedImageDatastore([inputSize(1:2) 1], imdsTest,
'ColorPreprocessing', 'gray2rgb');

% Extract features from the training and testing data
layer = 'pool10'; % You can choose a different layer if needed
featuresTrain = activations(net, augimdsTrain, layer, 'OutputAs',
'rows');
featuresTest = activations(net, augimdsTest, layer, 'OutputAs', 'rows');

% Save the extracted features to Excel
trainFeaturesFile = 'train_features.xlsx';
writematrix(featuresTrain, trainFeaturesFile, 'Sheet', 1);
disp(['Fitur dari data latih telah disimpan ke file: ',
trainFeaturesFile]);

testFeaturesFile = 'test_features.xlsx';
writematrix(featuresTest, testFeaturesFile, 'Sheet', 1);
disp(['Fitur dari data uji telah disimpan ke file: ',
testFeaturesFile]);

% Save the extracted features to MAT files
save('features.mat', 'featuresTrain', 'featuresTest');
disp('Fitur dari data latih dan uji telah disimpan ke file MAT:
features.mat');

% Get labels for training and testing data
labelsTrain = imdsTrain.Labels;
```

```
labelsTest = imdsTest.Labels;

% Apply PCA to reduce the dimensionality by 50%
numComponents = round(size(featuresTrain, 2) / 2);
[coeff, scoreTrain] = pca(featuresTrain, 'NumComponents',
numComponents);
scoreTest = featuresTest * coeff(:, 1:numComponents);

% Train the SVM classifier
classifier = fitcecoc(featuresTrain, labelsTrain);

% Predict the labels of the test set
predictedLabels = predict(classifier, featuresTest);

% Calculate the accuracy
accuracy = mean(predictedLabels == labelsTest);

% Confusion matrix
confMat = confusionmat(labelsTest, predictedLabels);

% Calculate precision, recall, F1 score, and support for each category
tp = diag(confMat);  % true positives
fp = sum(confMat, 1) - tp;  % false positives
fn = sum(confMat, 2) - tp;  % false negatives

precision = tp ./ (tp + fp);
recall = tp ./ (tp + fn);
F1 = 2 * (precision .* recall) ./ (precision + recall);
support = sum(confMat, 2);

% Calculate mean of all data
mean_precision = mean(precision);
mean_recall = mean(recall);
mean_F1 = mean(F1);

% Create a table to display the results
fprintf('Test Result:\n');
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
fprintf('\n');
fprintf('Results:\n');
fprintf('Category    Precision    Recall    F1 Score    Support\n');
for i = 1:length(categories)
    fprintf('  %s     %.2f%%     %.2f%%     %.2f%%     %d\n', ...
        categories{i}, precision(i)*100, recall(i)*100, F1(i)*100,
support(i));
end
fprintf('\n');
fprintf('Mean Results:\n');
fprintf('Precision: %.2f%%\n', mean_precision*100);
fprintf('Recall: %.2f%%\n', mean_recall*100);
fprintf('F1 Score: %.2f%%\n', mean_F1*100);

% Plot the confusion matrix
figure;
confusionchart(confMat, categories);
title('Confusion Matrix');
```

```
% Plot the SVM decision boundary and support vectors in 2D using PCA
figure;
gscatter(scoreTrain(:, 1), scoreTrain(:, 2), labelsTrain);
hold on;

% Create a grid for plotting the decision boundary
[x1Grid, x2Grid] = meshgrid(linspace(min(scoreTrain(:, 1)),
max(scoreTrain(:, 1)), 100), ...
    linspace(min(scoreTrain(:, 2)), max(scoreTrain(:, 2)), 100));
xGrid = [x1Grid(:), x2Grid(:)];

% Predict scores over the grid
[~, scores] = predict(classifier, xGrid * coeff(:, 1:2)');

% Plot the decision boundary
contour(x1Grid, x2Grid, reshape(scores(:, 2), size(x1Grid)), [0 0],
'k');
title('PCA Visualization with SVM Decision Boundary');
legend(categories, 'Location', 'Best');
hold off;

% Plot some sample images with their predicted labels
figure;
idx = randperm(numel(imdsTest.Files), 9);
for i = 1:9
    subplot(3, 3, i);
    img = readimage(imdsTest, idx(i));
    imshow(img);
    label = predictedLabels(idx(i));
    title(sprintf('Image %d: Predicted: %s', idx(i), char(label)));
end
```

CNN & PCA

```
clc;
% Directory containing the images
imageFolder = 'D:\Skripsi\img_test';

% Categories for classification
categories = {'matang', 'busuk', 'mentah'};

% Image properties (grayscale image)
imageSize = [100 100];
numComponents = 50; % Number of PCA components to keep

% Load images from folders
imds = imageDatastore(fullfile(imageFolder, categories), ...
    'LabelSource', 'foldernames', 'IncludeSubfolders', true, ...
    'FileExtensions', '.png');

% Convert images to grayscale and resize them
imds.ReadFcn = @(loc) imresize(rgb2gray(imread(loc)), imageSize);
```

```
% --- Plot Eigenvalues for One Image from Each Category ---
figure;
for i = 1:numel(categories)
    imdsCategory = subset(imds, imds.Labels == categories{i});
    img = readimage(imdsCategory, 1); % Take the first image from each
category
    imgReshaped = reshape(img, 1, []); % Reshape to a row vector

    % Perform PCA on the single image
    [~, ~, latent] = pca(double(imgReshaped), 'Centered', false);

    % Display eigenvalues
    subplot(1, numel(categories), i);
    plot(latent, '-o');
    title(['Eigenvalues: ', categories{i}]);
    xlabel('Component Number');
    ylabel('Eigenvalue');
end
sgtitle('Eigenvalues of PCA for One Image from Each Category');

% --- PCA on All Images ---
numImages = numel(imds.Files);
dataMatrix = zeros(numImages, prod(imageSize));
labels = imds.Labels; % Store labels

% Read and reshape all images
for i = 1:numImages
    img = readimage(imds, i);
    imgReshaped = reshape(img, 1, []);
    dataMatrix(i, :) = imgReshaped;
end

% Perform PCA on the entire dataset
[~, score, ~] = pca(double(dataMatrix), 'Centered', false);

% --- Plot the First Two Principal Components with Category Boundaries -
--
figure;

% Colors for different categories
colors = [1 0 0; 0 1 0; 0 0 1]; % Red, Green, Blue

% Plot each category with different colors
hold on;
for i = 1:numel(categories)
    categoryIndices = labels == categories{i};
    scatter(score(categoryIndices, 1), score(categoryIndices, 2), 36,
colors(i, :), 'filled', 'DisplayName', categories{i});
end

% Optionally, you can fit a linear boundary or decision boundary based
on PCA components
% This is a simple example, more complex methods like SVM or logistic
regression can be used

% Add legend and labels
legend('show');
```

```
title('PCA: First Two Principal Components with Category Boundaries');
xlabel('1st Principal Component');
ylabel('2nd Principal Component');

% Adjust plot limits and add grid for better visualization
xlim([min(score(:, 1)) - 1, max(score(:, 1)) + 1]);
ylim([min(score(:, 2)) - 1, max(score(:, 2)) + 1]);
grid on;
hold off;



% Split data into training (70%) and testing (30%)
[imdsTrain, imdsTest] = splitEachLabel(imds, 0.8, 'randomized');

% Apply PCA to training and testing data
[trainFeatures, trainLabels] = applyPCA(imdsTrain, imageSize,
numComponents);
[testFeatures, testLabels] = applyPCA(imdsTest, imageSize,
numComponents);

% Reshape the PCA features to 4D for CNN input
trainFeatures = reshape(trainFeatures', [1 1 numComponents
size(trainFeatures, 1)]);
testFeatures = reshape(testFeatures', [1 1 numComponents
size(testFeatures, 1)]);

% Ensure the labels are categorical
trainLabels = categorical(trainLabels);
testLabels = categorical(testLabels);

% Define the CNN architecture for 1D input
layers = [
    imageInputLayer([1 1 numComponents], 'Normalization', 'none')

    convolution2dLayer(1, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(1, 'Stride', 1)

    convolution2dLayer(1, 64, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(1, 'Stride', 1)

    convolution2dLayer(1, 128, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(1, 'Stride', 1)

    fullyConnectedLayer(numel(categories))
    softmaxLayer
    classificationLayer];

% Define path for log file
logFilePath = 'D:\Skripsi\training_progress.txt';
```

```
% Open log file for writing
fileID = fopen(logFilePath, 'w');
fprintf(fileID, 'Training Progress Log\n');
fprintf(fileID, '====================\n');
fclose(fileID);

% Set training options
options = trainingOptions('adam', ...
    'InitialLearnRate', 1e-4, ...
    'MaxEpochs', 20, ...
    'MiniBatchSize', 32, ...
    'ValidationData', {testFeatures, testLabels}, ...
    'ValidationFrequency', 30, ...
    'Verbose', true, ...
    'Plots', 'training-progress', ...
    'OutputFcn', @(info) writeProgressToLog(info, logFilePath));

% Train the CNN
net = trainNetwork(trainFeatures, trainLabels, layers, options);

% Predict on the test set
YPred = classify(net, testFeatures);
YTest = testLabels;

% Calculate accuracy
accuracy = sum(YPred == YTest) / numel(YTest);
disp(['Test Accuracy: ', num2str(accuracy)]);

% Confusion matrix as a figure
figure;
confusionchart(YTest, YPred);
title('Confusion Matrix for Test Data with PCA');

% Classification report
classes = unique(YTest);
numClasses = numel(classes);

% Initialize arrays to store metrics for each class
precisions = zeros(numClasses, 1);
recalls = zeros(numClasses, 1);
f1Scores = zeros(numClasses, 1);

fprintf('Classification Report:\n');
fprintf('----------------------\n');
fprintf('%-10s | %-10s | %-10s | %-10s\n', 'Class', 'Precision',
'Recall', 'F1 Score');
fprintf('----------------------\n');

for i = 1:numClasses
    classLabel = classes(i);
    TP = sum((YPred == classLabel) & (YTest == classLabel));
    FP = sum((YPred == classLabel) & (YTest ~= classLabel));
    FN = sum((YPred ~= classLabel) & (YTest == classLabel));

    % Avoid division by zero
    if (TP + FP) == 0
        precision = 0;
```

```
    else
        precision = TP / (TP + FP);
    end

    if (TP + FN) == 0
        recall = 0;
    else
        recall = TP / (TP + FN);
    end

    if (precision + recall) == 0
        F1 = 0;
    else
        F1 = 2 * (precision * recall) / (precision + recall);
    end

    % Store metrics
    precisions(i) = precision;
    recalls(i) = recall;
    f1Scores(i) = F1;

    fprintf('%-10s | %-10.4f | %-10.4f | %-10.4f\n', classLabel,
precision, recall, F1);
end

fprintf('----------------------\n');
fprintf('Macro Average:\n');
macroPrecision = mean(precisions);
macroRecall = mean(recalls);
macroF1 = mean(f1Scores);
fprintf('%-10s | %-10.4f | %-10.4f | %-10.4f\n', 'Macro Avg',
macroPrecision, macroRecall, macroF1);


% Display some test images along with predicted and true labels
numImagesToShow = 9; % Number of images to display
figure;
for i = 1:numImagesToShow
    subplot(3, 3, i);
    img = readimage(imdsTest, i);
    imshow(img);

    % Get the true label and predicted label for the current image
    trueLabel = YTest(i);
    predictedLabel = YPred(i);

    % Display the true and predicted labels as the title
    title(['True: ', char(trueLabel), ', Pred: ',
char(predictedLabel)]);
end
sgtitle('Sample Test Images with True and Predicted Labels');

% Function to apply PCA and return features with labels
function [pcaFeatures, labels] = applyPCA(imds, imageSize,
numComponents)
    numImages = numel(imds.Files);
    dataMatrix = zeros(numImages, prod(imageSize));
```

```
    labels = imds.Labels; % Store labels

    for i = 1:numImages
        img = readimage(imds, i);
        imgReshaped = reshape(img, 1, []); % Reshape to a row vector
        dataMatrix(i, :) = imgReshaped;
    end

    % Perform PCA
    [coeff, score, ~] = pca(dataMatrix);

    % Reduce to specified number of components
    pcaFeatures = score(:, 1:numComponents);
end

% Function to write training progress to log file
function writeProgressToLog(info, logFilePath)
    fileID = fopen(logFilePath, 'a');
    if info.State == "start"
        fprintf(fileID, 'Starting Training...\n');
    elseif info.State == "epoch"
        fprintf(fileID, 'Epoch %d: Training Loss: %.4f, Validation Loss:
%.4f, Validation Accuracy: %.4f\n', ...
            info.Epoch, info.TrainingLoss, info.ValidationLoss,
info.ValidationAccuracy);
    elseif info.State == "done"
        fprintf(fileID, 'Training Complete.\n');
    end
    fclose(fileID);
end
```

SVM

```
clc;
% Define the main folder containing the images
mainFolder = 'D:\Skripsi\img_test';

% Define the categories
categories = {'matang', 'mentah', 'busuk'};

% Load the image data
imds = imageDatastore(fullfile(mainFolder, categories), 'LabelSource',
'foldernames');

% Convert images to grayscale
imds.ReadFcn = @(filename) rgb2gray(imread(filename));

% Split the data into training (70%) and testing (30%)
[imdsTrain, imdsTest] = splitEachLabel(imds, 0.8, 'randomized');

% Extract features using a pre-trained network (SqueezeNet)
net = squeezenet;
inputSize = net.Layers(1).InputSize;

% Augment data to match the input size of the network
```

```
augimdsTrain = augmentedImageDatastore([inputSize(1:2) 1], imdsTrain,
'ColorPreprocessing', 'gray2rgb');
augimdsTest = augmentedImageDatastore([inputSize(1:2) 1], imdsTest,
'ColorPreprocessing', 'gray2rgb');

% Extract features from the training and testing data
layer = 'pool10'; % You can choose a different layer if needed
featuresTrain = activations(net, augimdsTrain, layer, 'OutputAs',
'rows');
featuresTest = activations(net, augimdsTest, layer, 'OutputAs', 'rows');

% Get labels for training and testing data
labelsTrain = imdsTrain.Labels;
labelsTest = imdsTest.Labels;

% Train the SVM classifier
classifier = fitcecoc(featuresTrain, labelsTrain);

% Predict the labels of the test set
predictedLabels = predict(classifier, featuresTest);

% Calculate the accuracy
accuracy = mean(predictedLabels == labelsTest);
fprintf('Accuracy: %.2f%%\n', accuracy * 100);

% Confusion matrix
confMat = confusionmat(labelsTest, predictedLabels);

% Plot the confusion matrix
figure;
confusionchart(confMat, categories);
title('Confusion Matrix');

% Plot some sample images with their true and predicted labels
figure;
idx = randperm(numel(imdsTest.Files), 9);
for i = 1:9
    subplot(3, 3, i);
    img = readimage(imdsTest, idx(i));
    trueLabel = labelsTest(idx(i));
    predLabel = predictedLabels(idx(i));
    imshow(img);
    % Display the true and predicted labels on the image
    hold on;
    text(10, 10, sprintf('True: %s', char(trueLabel)), 'Color',
'yellow', 'FontSize', 12, 'FontWeight', 'bold');
    text(10, 30, sprintf('Pred: %s', char(predLabel)), 'Color', 'red',
'FontSize', 12, 'FontWeight', 'bold');
    hold off;
end

% Display one specific image and its pixel values
figure;
imgIdx = 1; % Index of the image to display
img = readimage(imdsTest, imgIdx);
imgFilename = imdsTest.Files{imgIdx}; % Get the filename
imshow(img);
```

```
title(sprintf('Image: %s', imgFilename));

% Get the image dimensions
[imgHeight, imgWidth] = size(img);

% Define the region for pixel value display
region = [50, 50, 100, 100]; % [x, y, width, height]

% Ensure the region does not exceed the image bounds
x1 = max(1, region(1));
y1 = max(1, region(2));
x2 = min(imgWidth, region(1) + region(3) - 1);
y2 = min(imgHeight, region(2) + region(4) - 1);

% Extract the pixel values for the selected region
pixelRegion = img(y1:y2, x1:x2);

% Convert the pixel values to a table
[row, col] = size(pixelRegion);
pixelTable = array2table(pixelRegion);

% Define the filename to save the table
filename = 'pixel_values.csv';

% Write the table to a CSV file
writetable(pixelTable, filename);

% Display confirmation
fprintf('Pixel values saved to %s\n', filename);

% Classification report with mean F1 score, precision, and recall
report = classificationReport(labelsTest, predictedLabels, 'Categories',
categories);
disp(report);

% Accuracy process graph
% Assuming you have a history of accuracies over epochs or iterations
% Example placeholder data for accuracy
epochs = 1:10; % Replace with actual epoch numbers
trainAccuracy = rand(1, 10) * 0.5 + 0.5; % Replace with actual training
accuracies
testAccuracy = rand(1, 10) * 0.5 + 0.5; % Replace with actual testing
accuracies

% Plot training and testing accuracy over epochs
figure;
plot(epochs, trainAccuracy, '-o', 'DisplayName', 'Train Accuracy');
hold on;
plot(epochs, testAccuracy, '-o', 'DisplayName', 'Test Accuracy');
xlabel('Epochs');
ylabel('Accuracy');
title('Training and Testing Accuracy Over Epochs');
legend('show');
grid on;

% Local function for classification report with mean F1 score,
precision, and recall
```

```
function report = classificationReport(trueLabels, predictedLabels,
varargin)
    % Get unique labels
    labels = unique(trueLabels);

    % Initialize report string
    report = sprintf('Classification Report:\n\n');
    report = [report sprintf('%-10s %-10s %-10s %-10s\n', 'Class',
'Precision', 'Recall', 'F1-Score')];

    % Initialize sums for mean calculation
    sumPrecision = 0;
    sumRecall = 0;
    sumF1Score = 0;

    % Calculate metrics for each label
    for i = 1:length(labels)
        label = labels(i);

        % True positive, false positive, false negative, true negative
        tp = sum((trueLabels == label) & (predictedLabels == label));
        fp = sum((trueLabels ~= label) & (predictedLabels == label));
        fn = sum((trueLabels == label) & (predictedLabels ~= label));
        tn = sum((trueLabels ~= label) & (predictedLabels ~= label));

        % Precision, recall, F1 score
        precision = tp / (tp + fp);
        recall = tp / (tp + fn);
        f1Score = 2 * (precision * recall) / (precision + recall);

        % Handle NaNs
        if isnan(precision), precision = 0; end
        if isnan(recall), recall = 0; end
        if isnan(f1Score), f1Score = 0; end

        % Append to report
        report = [report sprintf('%-10s %-10.2f %-10.2f %-10.2f\n',
char(label), precision, recall, f1Score)];

        % Add to sums for mean calculation
        sumPrecision = sumPrecision + precision;
        sumRecall = sumRecall + recall;
        sumF1Score = sumF1Score + f1Score;
    end

    % Calculate means
    meanPrecision = sumPrecision / length(labels);
    meanRecall = sumRecall / length(labels);
    meanF1Score = sumF1Score / length(labels);

    % Append means to report
    report = [report sprintf('\n%-10s %-10.2f %-10.2f %-10.2f\n',
'Mean', meanPrecision, meanRecall, meanF1Score)];
end
```

CNN

```
clc;
% Directory containing the images
imageFolder = 'D:\Skripsi\img_test';

% Categories for classification
categories = {'matang', 'busuk', 'mentah'};

% Image properties (grayscale image)
imageSize = [100 100];

% Load images from folders
imds = imageDatastore(fullfile(imageFolder, categories), ...
    'LabelSource', 'foldernames', 'IncludeSubfolders', true, ...
    'FileExtensions', '.png');

% Convert images to grayscale and resize them
imds.ReadFcn = @(loc) imresize(rgb2gray(imread(loc)), imageSize);

% Split data into training (70%) and testing (30%)
[imdsTrain, imdsTest] = splitEachLabel(imds, 0.8, 'randomized');

% Read and preprocess training and testing data
numTrainImages = numel(imdsTrain.Files);
numTestImages = numel(imdsTest.Files);

% Prepare data for CNN (reshape to fit the CNN input layer)
trainData = zeros([imageSize 1 numTrainImages], 'single');
testData = zeros([imageSize 1 numTestImages], 'single');

for i = 1:numTrainImages
    img = readimage(imdsTrain, i);
    trainData(:, :, :, i) = img;
end

for i = 1:numTestImages
    img = readimage(imdsTest, i);
    testData(:, :, :, i) = img;
end

% Ensure the labels are categorical
trainLabels = categorical(imdsTrain.Labels);
testLabels = categorical(imdsTest.Labels);

% Define the CNN architecture
layers = [
    imageInputLayer([100 100 1], 'Normalization', 'none')

    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)

    fullyConnectedLayer(numel(categories))
    softmaxLayer
    classificationLayer];

% Define path for log file
logFilePath = 'D:\Skripsi\training_progress.txt';

% Open log file for writing
fileID = fopen(logFilePath, 'w');
fprintf(fileID, 'Training Progress Log\n');
fprintf(fileID, '======================\n');
fclose(fileID);

% Set training options
options = trainingOptions('adam', ...
    'InitialLearnRate', 1e-4, ...
    'MaxEpochs', 20, ...
    'MiniBatchSize', 32, ...
    'ValidationData', {testData, testLabels}, ...
    'ValidationFrequency', 30, ...
    'Verbose', true, ...
    'Plots', 'training-progress', ...
    'OutputFcn', @(info) writeProgressToLog(info, logFilePath));

% Train the CNN
net = trainNetwork(trainData, trainLabels, layers, options);

% Predict on the test set
YPred = classify(net, testData);
YTest = testLabels;

% Calculate accuracy
accuracy = sum(YPred == YTest) / numel(YTest);
disp(['Test Accuracy: ', num2str(accuracy)]);

% Confusion matrix as a figure
figure;
confusionchart(YTest, YPred);
title('Confusion Matrix for Test Data');

% Classification report
classes = unique(YTest);
numClasses = numel(classes);

% Initialize arrays to store metrics for each class
precisions = zeros(numClasses, 1);
recalls = zeros(numClasses, 1);
f1Scores = zeros(numClasses, 1);

fprintf('Classification Report:\n');
```

```
fprintf('---------------------\n');
fprintf('%-10s | %-10s | %-10s | %-10s\n', 'Class', 'Precision',
'Recall', 'F1 Score');
fprintf('---------------------\n');

for i = 1:numClasses
    classLabel = classes(i);
    TP = sum((YPred == classLabel) & (YTest == classLabel));
    FP = sum((YPred == classLabel) & (YTest ~= classLabel));
    FN = sum((YPred ~= classLabel) & (YTest == classLabel));

    % Avoid division by zero
    if (TP + FP) == 0
        precision = 0;
    else
        precision = TP / (TP + FP);
    end

    if (TP + FN) == 0
        recall = 0;
    else
        recall = TP / (TP + FN);
    end

    if (precision + recall) == 0
        F1 = 0;
    else
        F1 = 2 * (precision * recall) / (precision + recall);
    end

    % Store metrics
    precisions(i) = precision;
    recalls(i) = recall;
    f1Scores(i) = F1;

    fprintf('%-10s | %-10.4f | %-10.4f | %-10.4f\n', classLabel,
precision, recall, F1);
end

fprintf('---------------------\n');
fprintf('Macro Average:\n');
macroPrecision = mean(precisions);
macroRecall = mean(recalls);
macroF1 = mean(f1Scores);
fprintf('%-10s | %-10.4f | %-10.4f | %-10.4f\n', 'Macro Avg',
macroPrecision, macroRecall, macroF1);

% Display some test images along with predicted and true labels
numImagesToShow = 9; % Number of images to display
figure;
for i = 1:numImagesToShow
    subplot(3, 3, i);
    img = readimage(imdsTest, i);
    imshow(img);

    % Get the true label and predicted label for the current image
    trueLabel = YTest(i);
```

```
    predictedLabel = YPred(i);

    % Display the true and predicted labels as the title
    title(['True: ', char(trueLabel), ', Pred: ',
char(predictedLabel)]);
end
sgtitle('Sample Test Images with True and Predicted Labels');

% Function to write training progress to log file
function writeProgressToLog(info, logFilePath)
    fileID = fopen(logFilePath, 'a');
    if info.State == "start"
        fprintf(fileID, 'Starting Training...\n');
    elseif info.State == "epoch"
        fprintf(fileID, 'Epoch %d: Training Loss: %.4f, Validation Loss:
%.4f, Validation Accuracy: %.4f\n', ...
            info.Epoch, info.TrainingLoss, info.ValidationLoss,
info.ValidationAccuracy);
    elseif info.State == "done"
        fprintf(fileID, 'Training Complete.\n');
    end
    fclose(fileID);
end
```