Code for Training the Model

```
%% Dataset Paths
newDataDir = ‘folderpath’;
newTrainDir = fullfile(newDataDir, 'train');
newValDir = fullfile(newDataDir, 'validation');

% Define Datastore
imdsTrainNew = imageDatastore(newTrainDir, 'IncludeSubfolders', true, 'LabelSource',
'foldernames');
imdsValNew = imageDatastore(newValDir, 'IncludeSubfolders', true, 'LabelSource',
'foldernames');

% Resizing images
inputSize = [224 224 3];
inputSizeinc = [299 299 3]; % For InceptionV3
augimdsTrainNew = augmentedImageDatastore(inputSize(1:2), imdsTrainNew);
augimdsValNew = augmentedImageDatastore(inputSize(1:2), imdsValNew);
augimdsTrainNewinc = augmentedImageDatastore(inputSizeinc(1:2), imdsTrainNew);
augimdsValNewinc = augmentedImageDatastore(inputSizeinc(1:2), imdsValNew);

numClasses = numel(unique(imdsTrainNew.Labels));

%% Loading Previously Trained Models
load('trainedEffNet.mat');
load('trainedGoogLeNet.mat');
load('trainedMobileNet.mat');
load('trainedInceptionV3.mat');


% Modifying EfficientNetB0
lgraphEffNet = layerGraph(trainedEffNet);
newEffNetLayers1 = [
    fullyConnectedLayer(numClasses, 'Name', 'new_fc_effnet')
    softmaxLayer('Name', 'new_softmax_effnet')
    classificationLayer('Name', 'new_output_effnet')];
lgraphEffNet = replaceLayer(lgraphEffNet, 'output_effnet', newEffNetLayers1);

% Modify GoogLeNet
lgraphGoogLeNet = layerGraph(trainedGoogLeNet);
newFcGoogLeNet = fullyConnectedLayer(numClasses, 'Name', 'new_fc_googlenet',
'WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10);
lgraphGoogLeNet = replaceLayer(lgraphGoogLeNet, 'fc_googlenet', newFcGoogLeNet);
newClassLayerGoogLeNet = classificationLayer('Name', 'new_output_googlenet');
lgraphGoogLeNet = replaceLayer(lgraphGoogLeNet, 'output_googlenet',
newClassLayerGoogLeNet);

% Modify MobileNetV2
lgraphMobileNet = layerGraph(trainedMobileNet);
newFcMobileNet = fullyConnectedLayer(numClasses, 'Name', 'new_fc_mobilenetv2',
'WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10);
lgraphMobileNet = replaceLayer(lgraphMobileNet, 'fc_mobilenetv2', newFcMobileNet);
newClassLayerMobileNet = classificationLayer('Name', 'new_output_mobilenetv2');
lgraphMobileNet = replaceLayer(lgraphMobileNet, 'output_mobilenetv2',
newClassLayerMobileNet);

% Modify InceptionV3
lgraphInceptionV3 = layerGraph(trainedInceptionV3);
```

```
newFcInceptionV3 = fullyConnectedLayer(numClasses, 'Name', 'new_fc_inceptionv3',
'WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10);
lgraphInceptionV3 = replaceLayer(lgraphInceptionV3, 'fc_inceptionv3', newFcInceptionV3);
newClassLayerInceptionV3 = classificationLayer('Name', 'new_output_inceptionv3');
lgraphInceptionV3 = replaceLayer(lgraphInceptionV3, 'output_inceptionv3',
newClassLayerInceptionV3);

%% Training Options
options = trainingOptions('adam', ...
    'InitialLearnRate', 2e-4, ...
    'MaxEpochs', 5, ...
    'MiniBatchSize', 16, ...
    'ValidationData', augimdsValNew, ...
    'Plots', 'training-progress', ...
    'Verbose', true);

%% Training Models

trainedEffNet2 = trainNetwork(augimdsTrainNew, lgraphEffNet, options);
trainedGoogLeNet2 = trainNetwork(augimdsTrainNew, lgraphGoogLeNet, options);
trainedMobileNet2 = trainNetwork(augimdsTrainNew, lgraphMobileNet, options);
options.ValidationData = augimdsValNewinc;
trainedInceptionV32 = trainNetwork(augimdsTrainNewinc, lgraphInceptionV3, options);

%% Saving New Trained Models
save('newEffNet1.mat', 'trainedEffNet2');
save('newGoogLeNet1.mat', 'trainedGoogLeNet2');
save('newMobileNet1.mat', 'trainedMobileNet2');
save('newInceptionV31.mat', 'trainedInceptionV32');
```

Code for Testing the Trained Model

```
% Set Test Path
testPath = 'folderpath';
testDatastore1 = imageDatastore(testPath, 'IncludeSubfolders', true, 'LabelSource',
'foldernames');

% Load Trained Models
load('newEffNet.mat', 'trainedEffNet1');
load('newGoogLeNet.mat', 'trainedGoogLeNet1');
load('newMobileNet.mat', 'trainedMobileNet1');
load('newInceptionV3.mat', 'trainedInceptionV31');

% Resizing for Different Models
inputSize224 = [224, 224, 3];
inputSize299 = [299, 299, 3];
testDatastore224 = augmentedImageDatastore(inputSize224, testDatastore1);
testDatastore299 = augmentedImageDatastore(inputSize299, testDatastore1);

%% Ensemble Prediction
% Get class labels
predsEffNet = classify(trainedEffNet1, testDatastore224);
predsGoogLeNet = classify(trainedGoogLeNet1, testDatastore224);
predsMobileNet = classify(trainedMobileNet1, testDatastore224);
predsInceptionV3 = classify(trainedInceptionV31, testDatastore299);

% Majority Voting for Final Prediction
```

```
finalPreds = mode([predsEffNet, predsGoogLeNet, predsMobileNet, predsInceptionV3], 2);

% Get True Labels
actualLabels = testDatastore1.Labels;

% Compute Accuracy
accuracyEffNet = sum(predsEffNet == actualLabels) / numel(actualLabels) * 100;
accuracyGoogLeNet = sum(predsGoogLeNet == actualLabels) / numel(actualLabels) * 100;
accuracyMobileNet = sum(predsMobileNet == actualLabels) / numel(actualLabels) * 100;
accuracyInceptionV3 = sum(predsInceptionV3 == actualLabels) / numel(actualLabels) * 100;
ensembleAccuracy = sum(finalPreds == actualLabels) / numel(actualLabels) * 100;

disp(['EffNet Accuracy: ', num2str(accuracyEffNet), '%']);
disp(['GoogLeNet Accuracy: ', num2str(accuracyGoogLeNet), '%']);
disp(['MobileNet Accuracy: ', num2str(accuracyMobileNet), '%']);
disp(['InceptionV3 Accuracy: ', num2str(accuracyInceptionV3), '%']);
disp(['Ensemble Model Accuracy: ', num2str(ensembleAccuracy), '%']);

%% Compute Sensitivity (Recall)
confMat = confusionmat(actualLabels, finalPreds);
numClasses = size(confMat, 1);
sensitivity = zeros(numClasses, 1);

for i = 1:numClasses
    TP = confMat(i, i);
    FN = sum(confMat(i, :)) - TP;
    sensitivity(i) = TP / (TP + FN);
end

% Display Sensitivity for Each Class
uniqueLabels = unique(actualLabels);
for i = 1:numClasses
    disp(['Sensitivity for Class ', char(uniqueLabels(i)), ': ', num2str(sensitivity(i)
* 100), '%']);
end

%%  Confusion Chart
figure;
confusionchart(actualLabels, finalPreds);
title('Confusion Matrix for Ensemble Model');
```

Combined Code for Testing Image using the Ensemble Model and CDR Calculation

```
testImagePath = ‘folderpath’;
inputImg = imread(testImagePath);

% Loading Trained Models
load('newEffNet.mat', 'trainedEffNet1');
load('newGoogLeNet.mat', 'trainedGoogLeNet1');
load('newMobileNet.mat', 'trainedMobileNet1');
load('newInceptionV3.mat', 'trainedInceptionV31');

% Resizing Image for Different Models
inputSize224 = [224, 224];
inputImgResized224 = imresize(inputImg, inputSize224);
inputSize299 = [299, 299];
```

```
inputImgResized299 = imresize(inputImg, inputSize299);

% Predictions from Each Model
[YPredEffNet, scoresEffNet] = classify(trainedEffNet1, inputImgResized224);
[YPredGoogLeNet, scoresGoogLeNet] = classify(trainedGoogLeNet1, inputImgResized224);
[YPredMobileNet, scoresMobileNet] = classify(trainedMobileNet1, inputImgResized224);
[YPredInceptionV3, scoresInceptionV3] = classify(trainedInceptionV31,
inputImgResized299);

% Ensemble Prediction (Average Scores)
ensembleScores = (scoresEffNet + scoresGoogLeNet + scoresMobileNet + scoresInceptionV3)
/ 4;
[~, ensembleIdx] = max(ensembleScores, [], 2);
YPredEnsemble = trainedEffNet1.Layers(end).Classes(ensembleIdx);
predictedClass = char(YPredEnsemble);
confidenceScore = max(ensembleScores) * 100;

disp('Performing CDR Calculation...');

% Converting to Grayscale
grayImg = rgb2gray(inputImg);

% Applying CLAHE for Contrast Enhancement
claheImg = adapthisteq(grayImg, 'ClipLimit', 0.005, 'NumTiles', [8 8]);
figure, imshow(claheImg); title('CLAHE Enhanced Image');

rect = drawrectangle();
position = round(rect.Position);

% Extracting cropping coordinates
x = position(1);
y = position(2);
width = position(3);
height = position(4);

% Crops the image
croppedImg = imcrop(claheImg, [x, y, width, height]);

[height1, width1, ~] = size(inputImg);
if (height1 == 256 && width1 == 256)
    % Applying Otsu's threshold for Optic Disc
    otsuLevelDisc = graythresh(croppedImg);
    bwDisc = imbinarize(croppedImg, otsuLevelDisc);
    bwDisc = bwareaopen(bwDisc, 50);
    bwDisc = imfill(bwDisc, 'holes'); % Filling small gaps

    % Applying Otsu's threshold for Optic Cup
    otsuLevelCup = graythresh(croppedImg(bwDisc)); % Uses masked region
    bwCup = imbinarize(croppedImg, otsuLevelCup);
    bwCup = bwareaopen(bwCup, 50); % Removes small objects
    bwDisc = imfill(bwDisc, 'holes'); % Filling small gaps

    % Fitting Circles to Optic Disc and Cup
    [xDisc, yDisc, rDisc] = fitCircleToBoundary(bwDisc);
    [xCup, yCup, rCup] = fitCircleToBoundary(bwCup);

    % Computes CDR (Cup-to-Disc Ratio)
    cdr = (rCup / rDisc)^2;
    fprintf('Cup-to-Disc Ratio (CDR): %.2f\n', cdr);
```

```
elseif (height1 == 2048 && width1 == 3072)
    % Applying Otsu's Threshold for Optic Disc
    otsuLevelDisc = graythresh(croppedImg);
    bwDisc = imbinarize(croppedImg, otsuLevelDisc);
    bwDisc = bwareaopen(bwDisc, 5000);
    bwDisc = imfill(bwDisc, 'holes');

    % Applying Otsu's Threshold for Optic Cup
    otsuLevelCup = graythresh(croppedImg(bwDisc));
    bwCup = imbinarize(croppedImg, otsuLevelCup);
    bwCup = bwareaopen(bwCup, 5000);
    bwCup = imfill(bwCup, 'holes');

    % Fitting Circles to Optic Disc and Cup
    [xDisc, yDisc, rDisc] = fitCircleToBoundary(bwDisc);
    [xCup, yCup, rCup] = fitCircleToBoundary(bwCup);

    % Computes CDR (Cup-to-Disc Ratio)
    cdr = (rCup / rDisc)^2;
    fprintf('Cup-to-Disc Ratio (CDR): %.2f\n', cdr);
end

% Function to Fit Circles
function [xc, yc, r] = fitCircleToBoundary(binaryMask)
    boundaries = bwboundaries(binaryMask);
    if isempty(boundaries)
        xc = NaN; yc = NaN; r = NaN;
        return;
    end

    largestRegion = bwconvhull(binaryMask);
    boundaries = bwboundaries(largestRegion);
    if isempty(boundaries)
        xc = NaN; yc = NaN; r = NaN;
        return;
    end

    % Extracts Boundary Points
    boundaryPoints = boundaries{1};
    x = boundaryPoints(:, 2);
    y = boundaryPoints(:, 1);

    % Finds Centroid
    stats = regionprops(largestRegion, 'Centroid');
    centroid = stats.Centroid;

    % Removes Outliers
    distances = sqrt((x - centroid(1)).^2 + (y - centroid(2)).^2);
    maxDist = median(distances) * 1.2;
    validIdx = distances <= maxDist;
    x = x(validIdx);
    y = y(validIdx);

    % Least-Squares Circle Fitting
    A = [x, y, ones(size(x))];
    B = -(x.^2 + y.^2);
    sol = pinv(A) * B;  % Solves for circle parameters
```

```
    xc = -0.5 * sol(1);
    yc = -0.5 * sol(2);
    r = sqrt((sol(1)^2 + sol(2)^2) / 4 - sol(3));
end

figure;
imshow(croppedImg); hold on;

% Drawing Fitted Circles
theta = linspace(0, 2*pi, 100);
plot(xDisc + rDisc*cos(theta), yDisc + rDisc*sin(theta), 'g', 'LineWidth', 2); % Optic
Disc
plot(xCup + rCup*cos(theta), yCup + rCup*sin(theta), 'b', 'LineWidth', 2); % Optic Cup

title('Fundus Image', 'FontSize', 14, 'Color', 'black');
fig = gcf;
fig.Position(4) = fig.Position(4) + 100;
textY = 0.02;
if cdr < 0.3
    cdrPrediction = 'Normal';
elseif cdr >= 0.3 && cdr < 0.4
    cdrPrediction = 'Suspicious';
else
    cdrPrediction = 'Glaucoma';
end

uicontrol('Style', 'text', 'String', ['Ensemble Prediction: ', predictedClass], ...
          'FontSize', 12, 'ForegroundColor', 'red', 'BackgroundColor', 'white', ...
          'Units', 'normalized', 'Position', [0.35, textY, 0.3, 0.05]);

uicontrol('Style', 'text', 'String', ['CDR Value: ', num2str(cdr, '%.2f')], ...
          'FontSize', 12, 'ForegroundColor', 'red', 'BackgroundColor', 'white', ...
          'Units', 'normalized', 'Position', [0.35, textY + 0.10, 0.3, 0.05]);

uicontrol('Style', 'text', 'String', ['CDR-Based Prediction: ', cdrPrediction], ...
          'FontSize', 12, 'ForegroundColor', 'red', 'BackgroundColor', 'white', ...
          'Units', 'normalized', 'Position', [0.35, textY + 0.05, 0.3, 0.05]);
hold off;
```