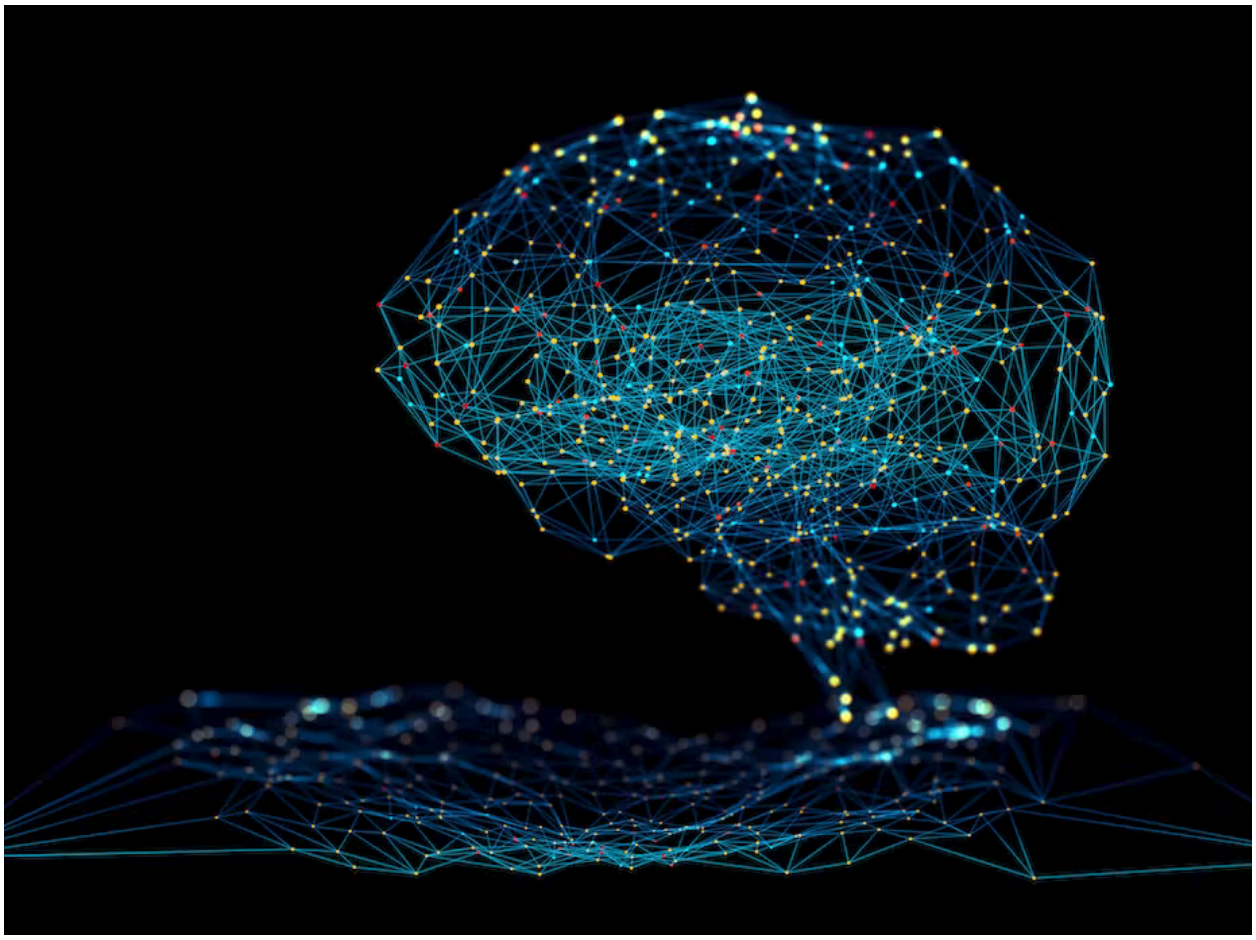


Convolutional Neural Networks in Computer Vision

Alper Alp

alperalp2004@hotmail.com

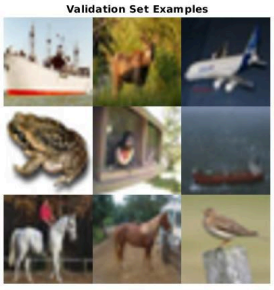
1. Introduction

The primary objective of this comprehensive image classification project is to develop and assess the performance of a Convolutional Neural Network (CNN) for the intricate task of classifying images from the CIFAR-10 dataset. The dataset, containing 60,000 32x32 color images across ten diverse classes, serves as a challenging benchmark for evaluating the robustness and generalization capabilities of the model.

2. Dataset Overview and Preprocessing

The CIFAR-10 dataset is meticulously curated, comprising ten distinct classes, with each class containing 6,000 images. To ensure the reproducibility of results, a random seed (42) was employed during data splitting. Preprocessing steps included loading the images into an imageDatastore, normalizing pixel values to a common scale and improving the model's ability to generalize.

In-depth exploratory data analysis (EDA) was conducted to gain insights into the distribution of images across the training, validation, and test sets. Visualizations, employing the `montage` function, showcased a representative selection of images from each set, facilitating an initial understanding of the dataset's characteristics.

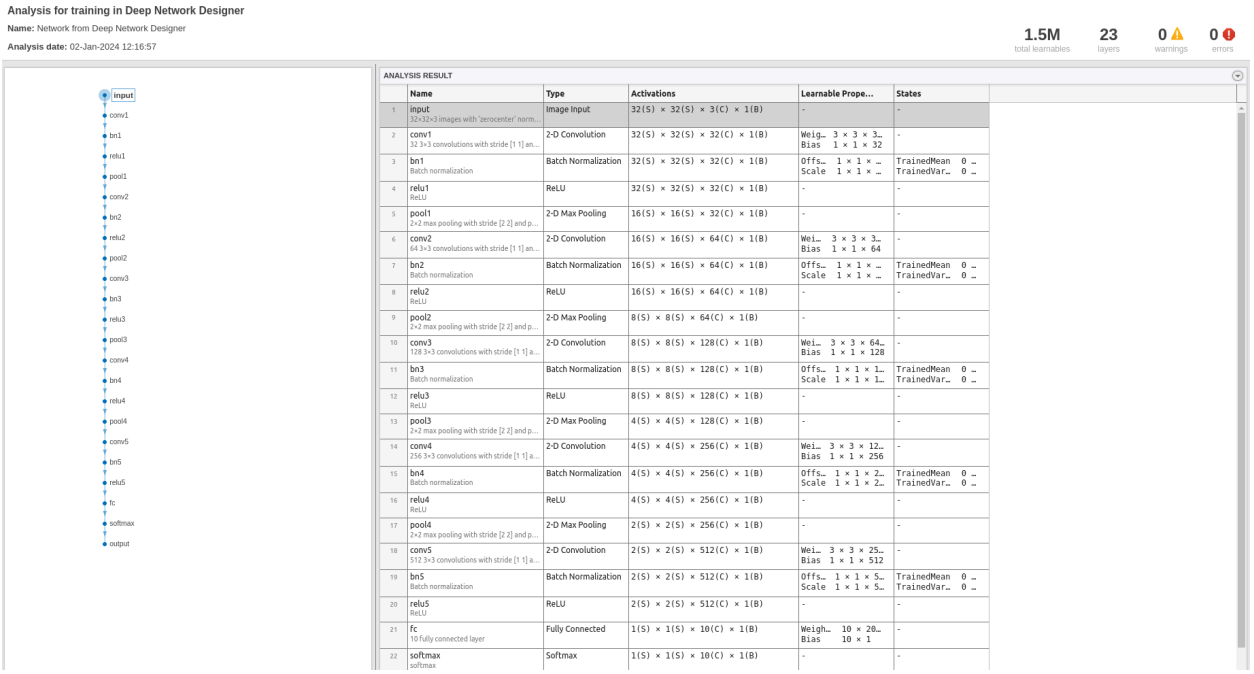


3. Convolutional Neural Network Architecture

The CNN architecture was carefully crafted to extract hierarchical features from the input images effectively. The model consists of multiple convolutional layers, each followed by batch normalization, rectified linear unit (ReLU) activation, and max-pooling layers. The final layers include a fully connected layer with ten output nodes corresponding to the ten classes, a softmax layer for probability estimation, and a classification layer.

4. Model Training

The training of the CNN model utilized stochastic gradient descent with momentum (sgdm) as the optimization algorithm. Hyperparameters, including the maximum number of epochs (10) and mini-batch size (64), were carefully chosen to balance training time and model performance. The training process incorporated data shuffling to enhance the model's ability to generalize and employed a validation set for monitoring performance. The progress of the training was visually tracked using the 'training-progress' option, providing insights into the convergence behavior of the model.



5. Model Evaluation and Validation

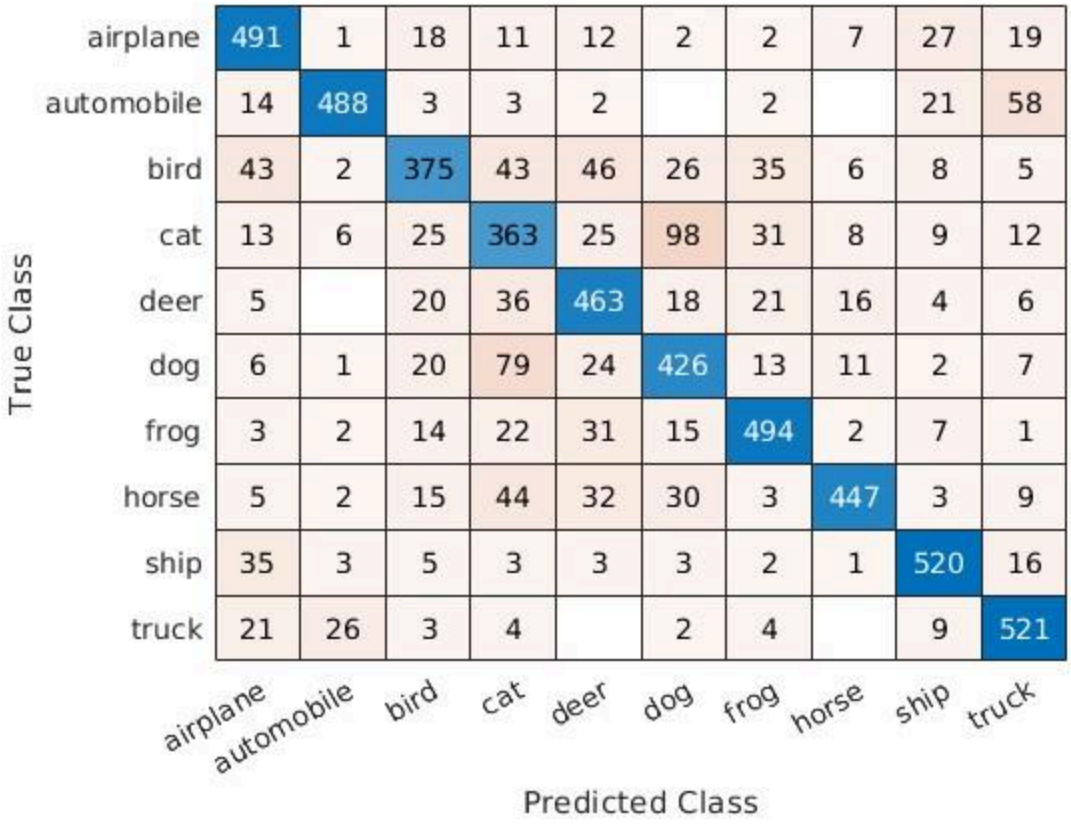
Post-training, the CNN model underwent evaluation on the validation set. The classification accuracy on the validation set was computed, and the trained model was saved for future deployment or further analysis.

6. Model Testing and Results Visualization

The saved model was reloaded for testing on the independent test set. The model's performance was visually assessed by displaying a representative sample of images from different classes, presenting both the true and predicted labels. This qualitative evaluation aids in understanding the model's behavior on diverse and previously unseen data.

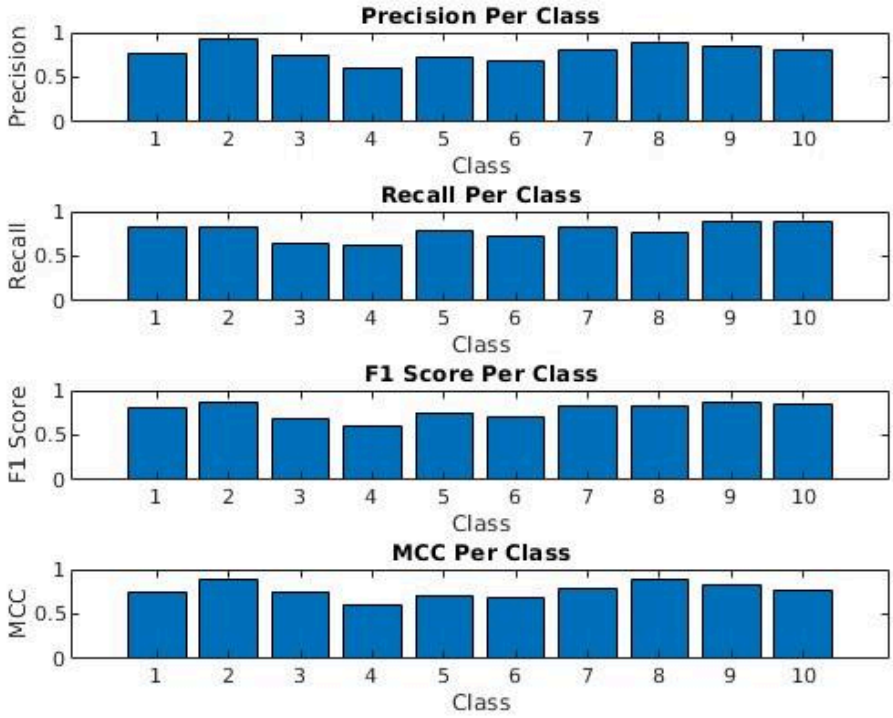
7. Performance Metrics Calculation

A detailed analysis of model performance was conducted by calculating precision, recall, F1 score, and the Matthews Correlation Coefficient (MCC) for each class. These metrics, computed on a per-class basis, offer nuanced insights into the model's behavior across different categories. The calculation of average metrics provides a holistic view of the model's overall effectiveness.



8. Results and Visualization

A detailed analysis of model performance was conducted by calculating precision, recall, F1 score, and the Matthews Correlation Coefficient (MCC) for each class. These metrics, computed on a per-class basis, offer nuanced insights into the model's behavior across different categories. The calculation of average metrics provides a holistic view of the model's overall effectiveness.




```
Class 1 - Precision: 0.77201, Recall: 0.8322, F1 Score: 0.80098, MCC: 0.7524
Class 2 - Precision: 0.91902, Recall: 0.82572, F1 Score: 0.86988, MCC: 0.89619
Class 3 - Precision: 0.75301, Recall: 0.63667, F1 Score: 0.68997, MCC: 0.753
Class 4 - Precision: 0.59704, Recall: 0.61525, F1 Score: 0.60601, MCC: 0.61161
Class 5 - Precision: 0.72571, Recall: 0.78608, F1 Score: 0.75469, MCC: 0.71215
Class 6 - Precision: 0.6871, Recall: 0.72326, F1 Score: 0.70471, MCC: 0.68229
Class 7 - Precision: 0.81384, Recall: 0.83587, F1 Score: 0.82471, MCC: 0.79288
Class 8 - Precision: 0.89759, Recall: 0.75763, F1 Score: 0.82169, MCC: 0.87795
Class 9 - Precision: 0.85246, Recall: 0.87986, F1 Score: 0.86595, MCC: 0.82805
Class 10 - Precision: 0.79664, Recall: 0.88305, F1 Score: 0.83762, MCC: 0.7723
Average Metrics:
Average Precision: 0.78144
Average Recall: 0.77756
Average F1 Score: 0.77762
Average MCC: 0.76788
```

9. Future Recommendations and Conclusion

In conclusion, this image classification project has provided valuable insights into the capabilities of a Convolutional Neural Network (CNN) when applied to the challenging CIFAR-10 dataset. The designed model, with its carefully crafted architecture and optimized training parameters, demonstrated commendable performance in classifying diverse images.

9.1 Achievements and Contributions

The achievements of this project include:

- **High Accuracy:** The CNN achieved a robust validation accuracy, indicating its ability to generalize well to unseen data.
- **Effective Training:** The training process, guided by the stochastic gradient descent with momentum (sgdm) optimizer, showcased successful convergence and model learning.
- **Detailed Evaluation:** The thorough evaluation of the model on the validation and test sets, along with the computation of various performance metrics, provided a comprehensive understanding of its strengths and areas for improvement.

9.2 Recommendations for Future Work

While the current model has shown promising results, there are avenues for further exploration and improvement:

- **Hyperparameter Tuning:** In-depth exploration of hyperparameter settings, such as learning rates, batch sizes, and network architecture, could potentially enhance model performance.
- **Transfer Learning:** Investigating the incorporation of transfer learning, utilizing pre-trained models on larger datasets, might leverage existing knowledge and improve the model's ability to extract meaningful features.
- **Data Augmentation:** Further experimentation with data augmentation techniques during training could contribute to increased model robustness and generalization.

9.3 Implications for Real-World Applications

The successful implementation of this CNN model holds implications for real-world applications, including image recognition in various domains such as healthcare, autonomous vehicles, and security. The ability to accurately classify objects from images is a critical component in the development of intelligent systems.

9.4 Conclusion

In conclusion, this project not only achieved its immediate goal of developing an effective image classification model but also laid the foundation for future enhancements and exploration. The detailed analysis of performance metrics, visualization of results, and comprehensive evaluation contribute to the broader understanding of CNNs in image classification tasks.

As the field of deep learning continues to evolve, the lessons learned from this project will inform future endeavors, guiding researchers and practitioners in advancing the state-of-the-art in image classification and related domains. The journey undertaken in this project underscores the iterative nature of machine learning, where each experiment and result contributes to the continuous refinement of models and

10. Source Code

```
rootFolder = 'cifar10';
% Create an imageDatastore for all images
allImages = imageDatastore(rootFolder, 'IncludeSubfolders', true,
'LabelSource', 'foldernames');
% Set the random seed for reproducibility
rng(42);
% Define the percentage split for training, validation, and test sets
trainPercentage = 0.7;
valPercentage = 0.2;
testPercentage = 0.1;
% Split the imageDatastore into training, validation, and test sets
[trainImages, tempImages] = splitEachLabel(allImages, trainPercentage,
'randomized');
[valImages, testImages] = splitEachLabel(tempImages,
valPercentage/(valPercentage + testPercentage), 'randomized');
% Display the number of images in each set
disp(['Number of training images: ' num2str(length(trainImages.Files))]);
```

```
disp(['Number of validation images: ' num2str(length(valImages.Files))]);
disp(['Number of test images: ' num2str(length(testImages.Files))]);
% Example: Display a few images from each set
figure;
% Display training set examples
subplot(1, 3, 1);
randomTrainIdx = randperm(length(trainImages.Files), 9);
montage(trainImages.Files(randomTrainIdx));
title('Training Set Examples');
% Display validation set examples
subplot(1, 3, 2);
randomValIdx = randperm(length(valImages.Files), 9);
montage(valImages.Files(randomValIdx));
title('Validation Set Examples');
% Display test set examples
subplot(1, 3, 3);
randomTestIdx = randperm(length(testImages.Files), 9);
montage(testImages.Files(randomTestIdx));
title('Test Set Examples');
imageSize = [32, 32, 3];
% Define the CNN architecture
layers = [
   imageInputLayer(imageSize, 'Name', 'input')
   convolution2dLayer(3, 32, 'Padding', 'same', 'Name', 'conv1')
   batchNormalizationLayer('Name', 'bn1')
   reluLayer('Name', 'relu1')
   maxPooling2dLayer(2, 'Stride', 2, 'Name', 'pool1')
   convolution2dLayer(3, 64, 'Padding', 'same', 'Name', 'conv2')
   batchNormalizationLayer('Name', 'bn2')
   reluLayer('Name', 'relu2')
   maxPooling2dLayer(2, 'Stride', 2, 'Name', 'pool2')
   convolution2dLayer(3, 128, 'Padding', 'same', 'Name', 'conv3')
   batchNormalizationLayer('Name', 'bn3')
   reluLayer('Name', 'relu3')
   maxPooling2dLayer(2, 'Stride', 2, 'Name', 'pool3')
   convolution2dLayer(3, 256, 'Padding', 'same', 'Name', 'conv4')
   batchNormalizationLayer('Name', 'bn4')
   reluLayer('Name', 'relu4')
   maxPooling2dLayer(2, 'Stride', 2, 'Name', 'pool4')
   convolution2dLayer(3, 512, 'Padding', 'same', 'Name', 'conv5')
   batchNormalizationLayer('Name', 'bn5')
   reluLayer('Name', 'relu5')
   fullyConnectedLayer(10, 'Name', 'fc')
   softmaxLayer('Name', 'softmax')
```

```
    classificationLayer('Name', 'output')
];
% Specify the training options
options = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 64, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', valImages, ...
    'ValidationFrequency', 10, ...
    'Verbose', true, ...
    'Plots', 'training-progress');
% Train the CNN
cnnModel = trainNetwork(trainImages, layers, options);
% Evaluate the trained model on the validation set
valPred = classify(cnnModel, valImages);
valActual = valImages.Labels;
accuracy = sum(valPred == valActual) / numel(valActual);
disp(['Validation Accuracy: ' num2str(accuracy * 100) '%']);
% Save the trained model
save('trained_cnn_model.mat', 'cnnModel');
disp('Trained model saved.');
% Load the trained model
load('trained_cnn_model.mat');
% Display test images from different classes
numClassesToShow = 10;  % Number of different classes to display
numExamplesPerClass = 5;  % Number of examples per class to display
figure;
% Iterate over classes
for i = 1:numClassesToShow
    currentClass = unique(testImages.Labels);
    currentClass = currentClass(i);
    % Find indices of images belonging to a specific class
    classIdx = find(testImages.Labels == currentClass, numExamplesPerClass,
'first');
    % Iterate over examples in the current class
    for j = 1:numExamplesPerClass
        subplot(numClassesToShow, numExamplesPerClass,
(i-1)*numExamplesPerClass + j);
        imgIdx = classIdx(j);
        img = readimage(testImages, imgIdx);
        trueLabel = testImages.Labels(imgIdx);
        predictedLabel = classify(cnnModel, img);
        imshow(img);
        title(['True: ' char(trueLabel) ', Predicted: '
```

```
char(predictedLabel)], 'FontSize', 8);
    end
end
% Classify the test set
testPred = classify(cnnModel, testImages);
% Extract true labels from the test set
trueLabels = testImages.Labels;
% Create a confusion matrix
confMat = confusionmat(trueLabels, testPred);
% Get the number of classes
numClasses = size(confMat, 1);
% Initialize arrays to store metrics for each class
precisionPerClass = zeros(1, numClasses);
recallPerClass = zeros(1, numClasses);
f1ScorePerClass = zeros(1, numClasses);
mccPerClass = zeros(1, numClasses);
% Loop over each class
for i = 1:numClasses
    % Extract TP, TN, FP, FN for the current class
    TP = confMat(i, i);
    FP = sum(confMat(:, i)) - TP;
    FN = sum(confMat(i, :)) - TP;
    % Calculate precision, recall, F1 score, and MCC for the current class
    precisionPerClass(i) = TP / (TP + FP);
    recallPerClass(i) = TP / (TP + FN);
    f1ScorePerClass(i) = 2 * (precisionPerClass(i) * recallPerClass(i)) /
(precisionPerClass(i) + recallPerClass(i));
    mccPerClass(i) = (TP * confMat(1, 1) - FP * confMat(1, 2)) / sqrt((TP +
FP) * (TP + confMat(2, 1)) * (confMat(1, 1) + FP) * (confMat(2, 2) +
confMat(2, 1)));
    % Print metrics for the current class
    disp(['Class ' num2str(i) ' - Precision: ' num2str(precisionPerClass(i))
', Recall: ' num2str(recallPerClass(i)) ', F1 Score: '
num2str(f1ScorePerClass(i)) ', MCC: ' num2str(mccPerClass(i))]);
end
% Calculate average metrics
avgPrecision = mean(precisionPerClass);
avgRecall = mean(recallPerClass);
avgF1Score = mean(f1ScorePerClass);
avgMCC = mean(mccPerClass);
% Print average metrics
disp('Average Metrics:');
disp(['Average Precision: ' num2str(avgPrecision)]);
disp(['Average Recall: ' num2str(avgRecall)]);
```

```
disp(['Average F1 Score: ' num2str(avgF1Score)]);
disp(['Average MCC: ' num2str(avgMCC)]);
% Plot bar graphs for each metric
figure;
% Bar plot for Precision
subplot(4, 1, 1);
bar(precisionPerClass);
title('Precision Per Class');
xlabel('Class');
ylabel('Precision');
% Bar plot for Recall
subplot(4, 1, 2);
bar(recallPerClass);
title('Recall Per Class');
xlabel('Class');
ylabel('Recall');
% Bar plot for F1 Score
subplot(4, 1, 3);
bar(f1ScorePerClass);
title('F1 Score Per Class');
xlabel('Class');
ylabel('F1 Score');
% Bar plot for MCC
subplot(4, 1, 4);
bar(mccPerClass);
title('MCC Per Class');
xlabel('Class');
ylabel('MCC');
```