Signal and Image Processing - Mod. 1

Report for the **July 2024** exam

Project members:

Tito Nicola Drugman, 502252, titonicola.drugman01@universitadipavia.it
Angelica Pagni, 505887, angelica.pagni01@universitadipavia.it

Introduction

The dataset is composed of 115 images with 39 underexposed, 34 overexposed and 42 correctly exposed. For collecting the images we used a smartphone and we adjusted the exposure by scrolling up or down. The correctly classified images were captured with the automatic exposure setting time of the smartphone. For practical reasons, we decided to create a folder for the dataset which was divided into three subfolders. The structure is the following:

```
dataset/
        normal/
                img1.jpg
                img2.jpg
                …
        overexposed/
                img1.jpg
                img2.jpg
                …
        underexposed/
                img1.jpg
                img2.jpg
                …
```

For the creation of the project we used a subset of the dataset to run our code faster and we gradually increased our dataset to reach the requirement imposed.

The code usually takes 15-17 minutes to complete.

First section: Classification

The first step was the classification part where we need to use low-level features extracted from images and their histograms to classify images into the three categories (overexposed, underexposed and correctly exposed). We first created two vectors: one which contains the image data that we will use for the classification and one vector for the labels. Then we created a nested loop to iterate on the subfolders and on the images to collect the features. We read the image and converted it to YCbCr color space. In this format, luminance information is stored as a single component (Y). We then create the histogram of the Y channel and we normalize it.

Finally we extract the features we use for the classification:
- **meanY**: average intensity value.
- **varY**: spread (variance) of intensity value.
- **skewY**: asymmetry (skewness) of the intensity distribution.
- **minBinCount**: number of pixels in the image that have the minimum value in the Y.
- **maxBinCount**: number of pixels in the image that have the maximum value in the Y.

Mean, Variance and Skewness can be useful metrics since they will tell us information about the distribution of the histogram. We expect that in underexposed images the distribution will be greatly shifted to the left, while for overexposed images to the right. Furthermore, minBinCount and maxBinCount are useful metrics since we expect for underexposed images to have a greater minBinCount and a smaller maxBinCount and the opposite for overexposed images. Finally we appended the extracted features and their corresponding label to *imageData* and *labels*. Each label corresponds to the exposure category of the image (1 for underexposed, 2 for correctly exposed, 3 for overexposed).

Train-test split
We divided the data in the train and test set, and we decided to use 80% of the images (to be more precise: of the feature of the image) on the train, and 20% on the test.

In this process, we initialize empty vectors for features (*XTrain, XTest*) and labels (*YTrain, YTest*). We then loop through each category to retrieve the indices of the data entries for each category and count the number of images. By multiplying the number of images by the split ratio (0.8) and rounding to the nearest whole number, we determine the number of images for the training set for each category. We shuffle the data to eliminate any potential bias from the original ordering, then assign the first part of the shuffled data to the training set and the remaining part to the test set. This ensures that 80% of the data from each category is used for training, resulting in the training set containing approximately 80% of the entire dataset.

Feature scaling
At first we decided to apply feature scaling on XTrain and to XTest, to our surprise, the results were worse in terms of correctly classified images than without the normalization. We applied a scaling only to minBinCount and maxBinCount, otherwise they would have a higher magnitude than the other features.

Classifiers

K-Nearest Neighbors (KNN)

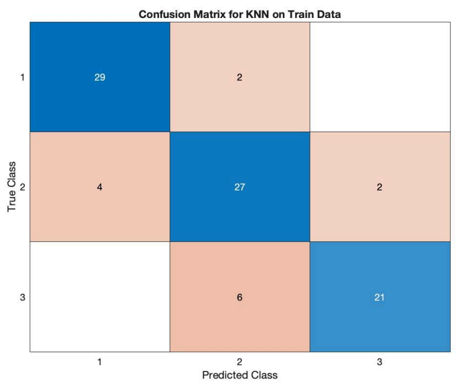
First we needed to find the optimal number of neighbors, our candidates were chosen in the range from one to twenty. For each value of k, a KNN model is created using the training data (XTrain and YTrain) and we compute the fold loss. Then an elbow plot is generated to visualize the error rates for different k values and to find the optimal number of neighbors.

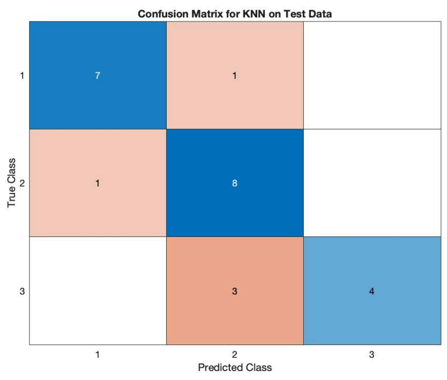
We created a 10-fold cross-validation partition for the training data and we used the optimal number of neighbors we found before. With a 10-fold cross-validation the data is divided into 10 subsets. In each of the 10 iterations, 9 subsets are used for training, and 1 subset is used for validation. Then we compute the loss (error rate) for each fold individually, resulting in an array foldLosses that contains the error rates for each of the 10 folds. We use this to find the fold with the minimum loss (that is the best performance) and we extract the model trained on the best-performing fold.

Then we evaluate performance on the training data and test data. We compute the accuracy and we plot the confusion matrix.

Train Accuracy KNN: 84.6154%

Test Accuracy KNN: 79.1667%



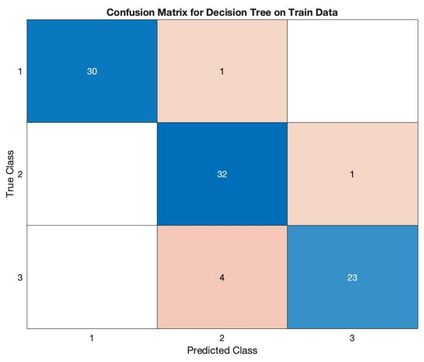


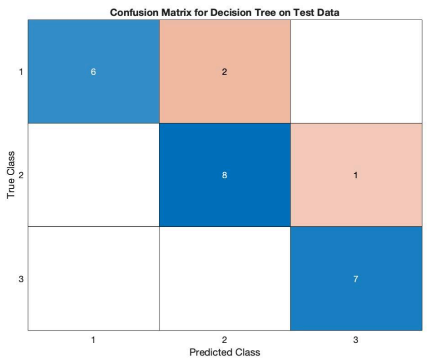
Decision Tree

We also decided to use a Decision Tree for classification. The goal is to find the best-performing Decision Tree model based on cross-validation performance, and then evaluate this model on both the train and test datasets. We compute the accuracy and we plot the confusion matrix. The test accuracy between the two classifiers was the same, but the confusion matrices were different.

Train Accuracy Decision Tree: 94.5055%

Test Accuracy Decision Tree: 87.5%





Second section: Gamma Correction

We first tried a brute-force approach in which we set a minimum and maximum value for the gamma (0.1 and 5 respectively with a 0.1 step) and then we tried to reach the values of the features of the correctly exposed images. Sadly this approach was not really suitable and the results were not particularly good. We then tried to refine the model by changing the step and the maximum value of the gamma, but the results remained almost unchanged. To decrease the gamma search space we even considered the fact that a gamma value less than 1 weights the mapping toward higher (brighter) output values and a gamma value of more than 1 weights output values toward lower (darker) output values, but this just speeded up the results without any improvement.

We decided to try a better solution and so firstly we calculated the target mean, variance, skew, and min/max bin count for normally exposed images. We took the correctly exposed images and converted to YCbCr color space and we extracted the Y channel. We then computed the statistics of the Y channel and we appended it to their proper list to compute their average.

Explanation of the Gamma Correction Code
We need to apply a gamma correction to all the images such that their exposure levels are adjusted to be correctly exposed according to the statistics calculated previously.

We create three functions:
applyGammaCorrection(img, gamma): a simple function that given an image and a gamma value it creates the gamma-corrected image.

compute_gamma_error(gamma, img, target_mean, target_variance, target_skew, target_minBinCount, target_maxBinCount): it is used to calculate the error between the gamma-corrected image statistics and the target statistics. It has different parameters: target_mean, target_variance, target_skew and min/maxBinCount The code first applies the gamma correction to the image and then it computes those statistics for the corrected image and it computes the weighted error. The weighted error is used to guide the optimization process to find the gamma value that minimizes these discrepancies. We tried different combinations of weights but the best one we found had very small weights for skewness and min/maxBinCount.

find_best_gamma(img, target_mean, target_variance, target_skew, target_minBinCount, target_maxBinCount, lower_bound, upper_bound): it is a function used to find the best gamma value that minimizes the error between corrected images statistics and the target statistics. This function uses MATLAB's *fmincon* function, which is an optimizer to find the gamma value that minimizes the error computed by *compute_gamma_error*. To speed up the search process we use a lower bound and upper bound for the gamma correction, which are defined later. We decided to save the images corrected with the gamma in a new folder, organized with the same structure of our dataset. Then we used a nested loop to iterate through each subfolder and each image in the dataset. We separated the channels YCbCr and we set different boundaries on the gamma value of the images.

- Underexposed images have a range allowing brightening, and so the gamma search space was between 0.1 and 1.
- Correctly exposed images have a tighter range around 1, suggesting minor adjustment.
- Overexposed images have a range allowing darkening, and so the gamma search space was greater than 1.

The code then called *find_best_gamma* to find the best gamma, apply the gamma correction thanks to *applyGammaCorrection* and recombine the Y corrected channel with Cb and Cr to finally convert the image back to RGB and save the corrected image.

Evaluate the gamma-corrected images

First we collected all the gamma-corrected images and their respective exposure category of each image. Then with a for loop we extract several key statistical features by converting it from RGB to YCbCr color space. We extracted the following features: MeanY, VarY, SkewY, MinBinCount and MaxBinCount and we aggregated them in *features_corrected*.

Results with KNN and Decision Tree
First we create *knnCounts* which is a vector used to count the number of images classified into each of the three categories. We again use a nested loop, the first one iterates through each subfolder and the other loop processes each image in the current subfolder. Each image is converted to YCbCr and we isolate the Y channel. Then we compute and normalize the histogram of the Y channel and we calculate the statistics. Then those statistics are compiled into a feature vector, which will be used for KNN classification.

This vector is fed into the pre-trained KNN classifier to predict the exposure category, we also count for the predicted category in knnCounts and for final checking we print each image's name and category to manually verifying how images are categorized by the model. We repeat this for the Decision Tree.

We then display a summary of the classification results for both KNN and Decision Tree.

KNN:
- Number of images classified as correctly exposed: 75
- Number of images classified as underexposed: 33
- Number of images classified as overexposed: 7

Decision Tree:
- Number of images classified as correctly exposed: 89
- Number of images classified as underexposed: 17
- Number of images classified as overexposed: 9

Third section: NIQE and entropy-kurtosis

NIQE
NIQE is a no-reference image quality metric that evaluates perceptual quality based on natural scene statistics, ideal for assessing images without needing a reference standard. We applied NIQE to original and gamma-corrected images to objectively measure improvement in perceptual quality. The NIQE function calculates a perceptual quality score that reflects the naturalness of images based on statistical regularities observed in natural images which are absent in unnatural ones. A smaller score indicates better perceptual quality. Finally the code counts how many images have a lower (improved) NIQE score after correction and it reports the total number of improved images and calculates the percentage of images where perceptual quality was enhanced post-correction.
- Number of images with improved NIQE: 45 out of 115
- Percentage of images with improved perceptual quality: 39.13%

Entropy-Kurtosis
Entropy measures the randomness or unpredictability in the image data, while kurtosis indicates the “tailedness” of the pixel values distribution. We iterate over the *correctly_classified* images to find the desired average entropy and the desired average kurtosis. We then iterate over each subfolder and compute the entropy and kurtosis of the original image and the gamma corrected one. The code outputs the name of each processed file along with its entropy and kurtosis values before and after gamma correction to provide immediate feedback on the effect of the correction process for each image.

Final plots
We plot the entropy values for each image after the gamma correction and a horizontal dashed line that represent the average desired entropy. We do the same for kurtosis. Both plots are important for understanding the results of the gamma correction with respect to the desired average values, helping determine if the gamma correction has standardized the images towards a desired state or if outliers and anomalies still persist. For checking the results we also plot an image with the respective Y histogram.