DEEP LEARNING

Assignment 01

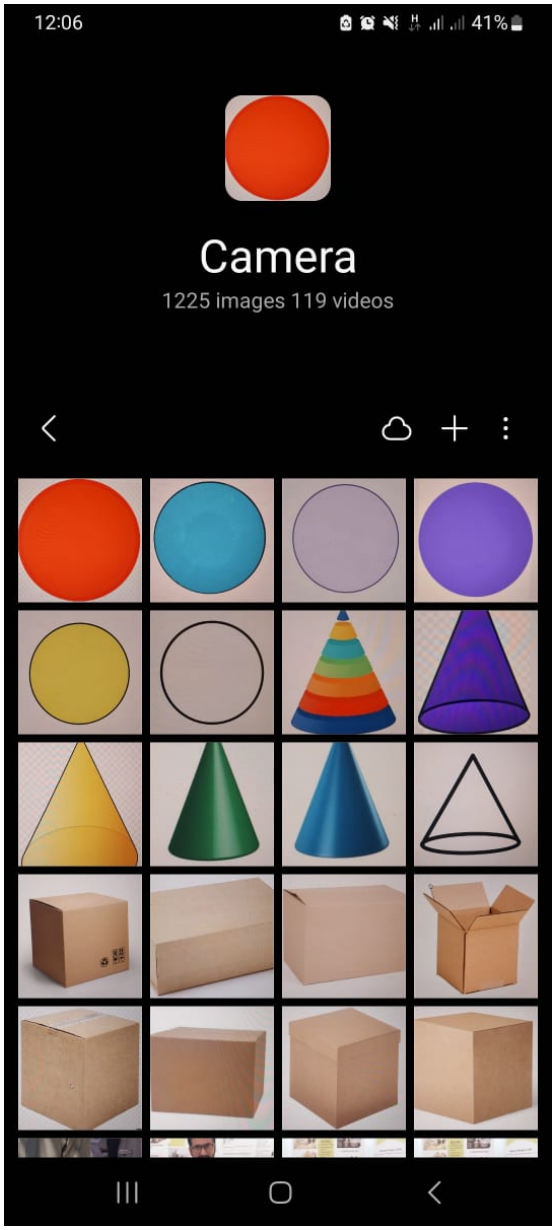
COURSE INSTRUCTOR
DR. JAWAD

RANA M. SAAD
MS-EE (AI&AS)
CMS: 400363

Report on Image Classification using K-Nearest Neighbors (KNN)

Introduction

This report discusses a Python code that performs image classification using the K-Nearest Neighbors (KNN) algorithm. The code is designed to work with a dataset containing images of cones, boxes, and circles captured from a phone camera.



It loads these images, preprocesses them, splits them into training and testing sets, and applies KNN to predict the category of each image. Additionally, the code displays sample images from each category and generates a graph to visualize the relationship between the number of neighbors (k) and the accuracy of the KNN algorithm.

Code Overview

1. **Importing Libraries**:

   The code begins by importing necessary libraries, including OpenCV (**cv2**), operating system (**os**), NumPy (**numpy**), Matplotlib (**matplotlib.pyplot**), and the **Counter** class from the **collections** module.

2. **Data Loading and Preprocessing**:

   - The path to the dataset directory is specified in the **data_dir** variable.
   - Empty lists (**data** and **labels**) are initialized to store image data and corresponding labels.
   - The code iterates through subdirectories in the dataset folder, loads images (in the formats .jpg, .jpeg, .png, .gif), resizes them to 64x64 pixels, converts them to grayscale, and flattens them into 1D arrays. Image data and labels are appended to the respective lists.

3. **Data Conversion**:

   - NumPy arrays **X** and **y** are created from the **data** and **labels** lists, respectively, to prepare the data for KNN classification.

4. **KNN Implementation**:

   The code defines a function **knn_predict** that implements KNN classification. It calculates distances between test samples and training data points, identifies the k nearest neighbors, and predicts the most common label among those neighbors.

5. **Data Splitting**:

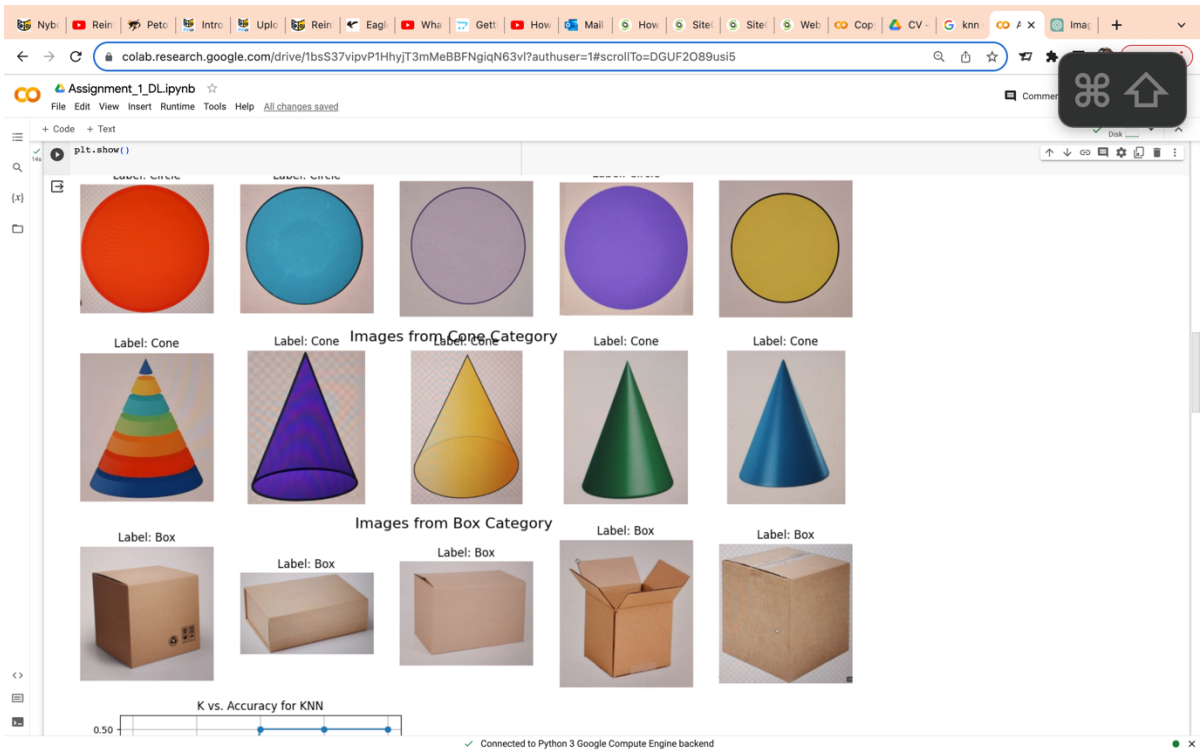
   - The dataset is split into training and testing sets using a specified split ratio (**split_ratio**).
   - **X_train**, **X_test**, **y_train**, and **y_test** are created to store the training and testing data and labels.

6. **KNN Accuracy Evaluation**:

   - The code defines a range of k values (**k_values**) to test the KNN algorithm's performance.
   - It calculates accuracy for each value of k by comparing predicted labels (**y_pred**) to the true labels (**y_test**).
   - The accuracies for different k values are stored in the **accuracies** list.

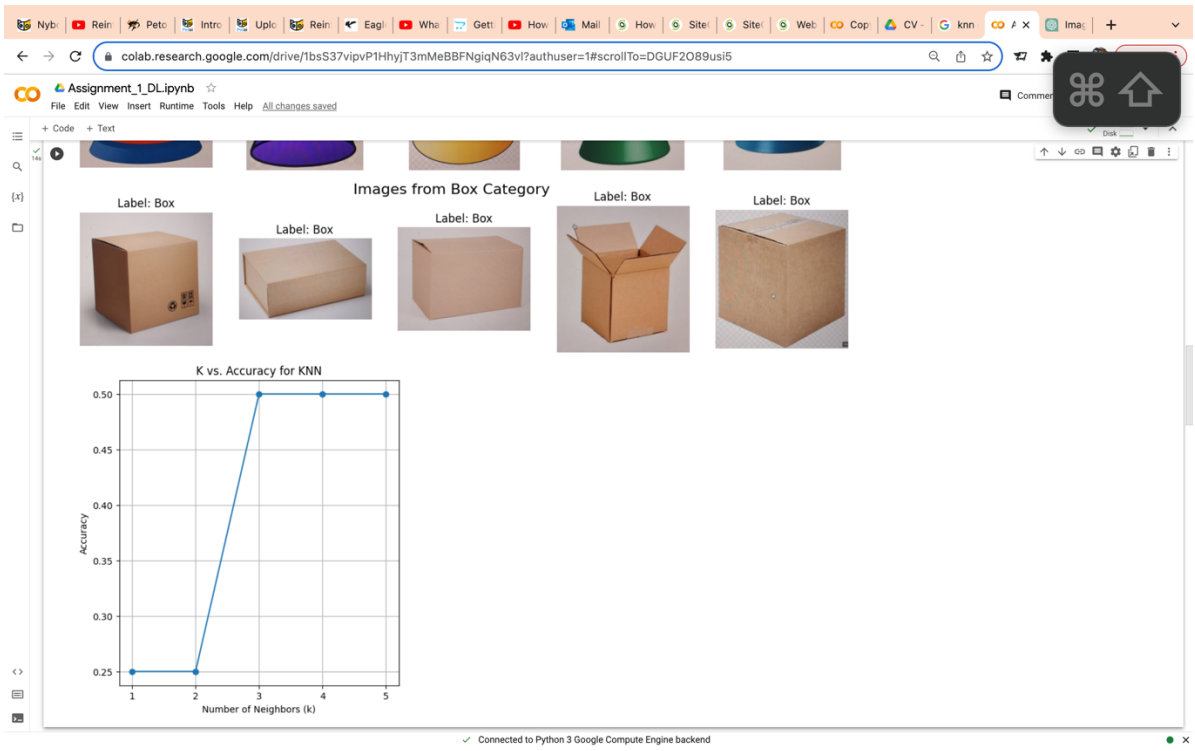
7. **Image Visualization**:

   The code displays sample images from each category separately using Matplotlib. It shows up to 5 images from each category along with their respective labels.



8. **Accuracy vs. K Visualization**:

   - The code creates a graph to visualize the relationship between the number of neighbors (k) from 1 to 5 and the accuracy of the KNN algorithm.
   - The x-axis represents the number of neighbors (k), and the y-axis represents accuracy.



Conclusion

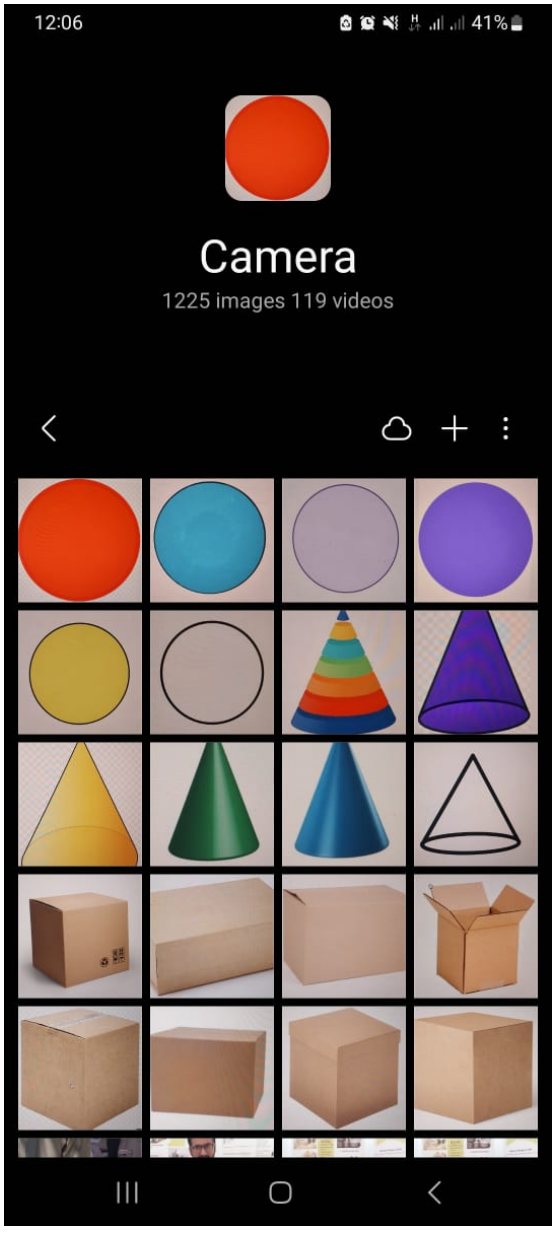
The provided Python code demonstrates a simple implementation of image classification using the K-Nearest Neighbors (KNN) algorithm. It loads a dataset containing images of cones, boxes, and circles, preprocesses the data, and evaluates the KNN algorithm's accuracy for different values of k. Additionally, it provides a visual representation of sample images from each category and a graph showing how the choice of k affects the classification accuracy.

This code can serve as a starting point for image classification tasks, and further enhancements can be made to improve classification performance, such as using more advanced feature extraction techniques and machine learning algorithms.

Report on Image Classification using SVM

Introduction

This report discusses a Python code that performs image classification using Support Vector Machines (SVM). The code uses a dataset consisting of images of cones, boxes, and circles captured with a phone camera.



It loads and preprocesses these images, splits them into training and testing sets, and trains an SVM classifier to predict the category of each image. The code also displays sample images from each category.

Code Overview

1. **Importing Libraries**:

   The code starts by importing the necessary libraries, including OpenCV (**cv2**), operating system (**os**), NumPy (**numpy**), and Matplotlib (**matplotlib.pyplot**).

2. **Data Loading and Preprocessing**:

   - The path to the dataset directory is defined in the **data_dir** variable.
   - Two empty lists, **data** and **labels**, are initialized to store image data and corresponding labels.
   - The code iterates through subdirectories in the dataset folder, loads images (in the formats .jpg, .jpeg, .png, .gif), resizes them to 64x64 pixels, converts them to grayscale, and flattens them into 1D arrays. Image data is stored in the **data** list, and integer-encoded labels are stored in the **labels** list.

3. **Data Conversion**:

   - NumPy arrays **X** and **y** are created from the **data** and **labels** lists, respectively, to prepare the data for SVM classification.
   - The labels are converted to integers using **dtype=int**.

4. **Data Splitting**:

   - The dataset is split into training and testing sets using a specified split ratio (**split_ratio**).
   - **X_train**, **X_test**, **y_train**, and **y_test** are created to store the training and testing data and labels.

5. **SVM Training Function**:

   The code defines a simplified linear SVM training function (**train_svm**). It uses stochastic gradient descent to update the weights and bias iteratively to minimize the hinge loss.

6. **Training the SVM**:

   - The SVM is trained by calling the **train_svm** function with the training data.
   - The resulting weights and bias are stored in the variables **weights** and **bias**.

7. **SVM Prediction Function**:

   The code defines a prediction function (**predict_svm**) to make predictions on the testing data.

8. **Making Predictions**:

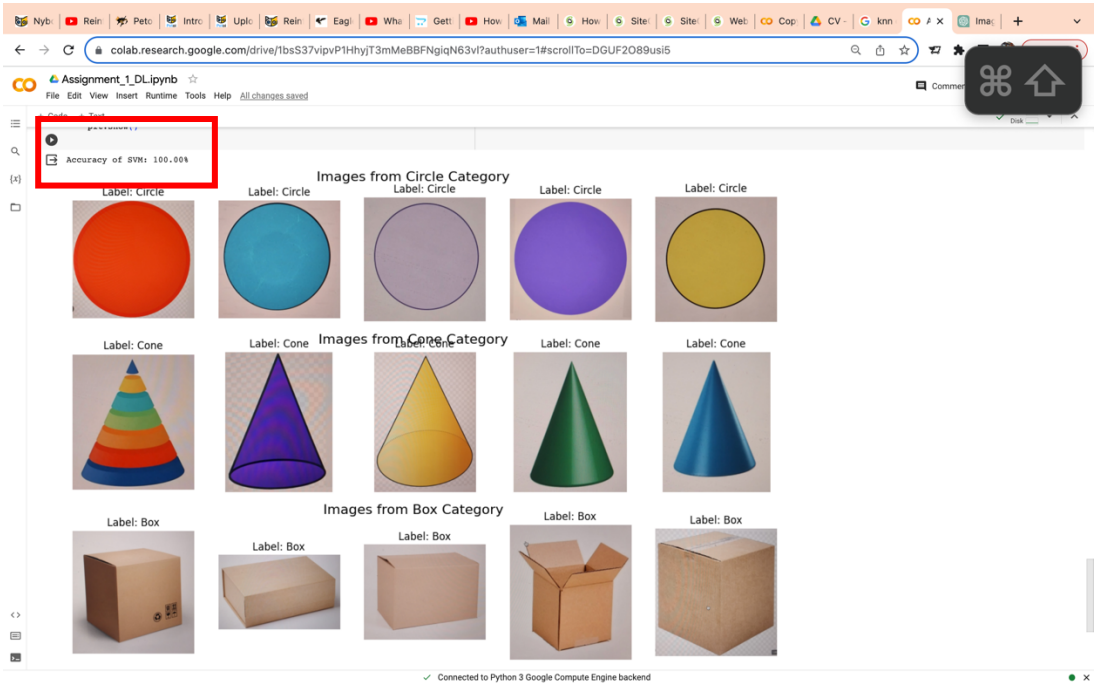
   - Predictions are made on the testing data by calling the **predict_svm** function with **X_test**, **weights**, and **bias**.
   - The predicted labels are stored in **y_pred**.

9. **Calculating Accuracy**:

   The code calculates the accuracy of the SVM classifier by comparing **y_pred** to **y_test**. The accuracy is then printed.

10. **Image Visualization**:

- The code displays sample images from each category using Matplotlib.
- Up to 5 images from each category are shown along with their respective labels.



Conclusion

The provided Python code demonstrates image classification using a simplified linear Support Vector Machine (SVM) on a dataset containing images of cones, boxes, and circles. It loads, preprocesses, and splits the data before training the SVM model. The accuracy of the SVM model is calculated and printed. Additionally, the code displays sample images from each category to provide a visual representation of the dataset.

This code can serve as a starting point for image classification tasks, and further improvements can be made by exploring different SVM kernels and tuning hyperparameters to enhance classification performance.