Computer Vision Challenge 2023: Documentation

Flower Dataset Challenge

GROUP MEMBERS:

Anmol Zehrah 20K-0199
Hamza Sameer Khan 20K-1744
Muhammad Anas Ahmed 20K-0237

Date Submitted ....December 10th, 2023
Instructors .... Dr. Muhammad Farrukh Shahid & Sir Sohail Ahmed

Contents

1 Introduction

Flowering plants exhibit a rich diversity of species, each characterized by unique visual features. Leveraging the power of machine learning and computer vision, this project embarks on the task of automating the classification of different types of flowers. The journey unfolds through key phases, from the meticulous curation and augmentation of a specialized dataset to the training of a ResNet model for accurate flower recognition. The project not only delves into the intricacies of classification but also addresses the vital aspect of object localization through specialized scripts. These scripts refine bounding box annotations, enhancing the model's precision in identifying flower regions within images. The evaluation process incorporates comprehensive metrics, and the testing phase extends the model's application to real-world scenarios, particularly in the context of video sequences. This synthesis of data science, machine learning, and computer vision encapsulates the essence of the flower classification project, offering valuable insights into the development of robust and adaptive machine learning systems.

2 Dataset Exploration

The code defines two functions, `explore_images` and `explore_video_frames`, to visually explore images within a specified folder and frames from a video file, respectively. Additionally, the code showcases the functionality by applying these functions to a flower dataset and a flower video.

The code serves as a visual exploration tool for images and video frames within a flower dataset. The `explore_images` function displays a grid of sample images from different flower classes found in the training folder of the dataset. Each subplot in the grid presents an image along with the corresponding flower class label. On the other hand, the `explore_video_frames` function extracts frames from a flower-related video file and displays a selection of eight frames evenly distributed throughout the video. The resulting visualizations provide a quick overview of the content in the dataset, aiding in understanding the diversity of flower classes and the visual characteristics of the video frames. The code concludes by applying these functions to a specific flower dataset and a flower video, offering a hands-on demonstration of the exploration tool.

3 Data Pre-Processing

3.1 Image Resizing & Normalization

The code defines a function called `preprocess_and_save` designed to resize and normalize pixel values of images in a training dataset. The function takes three parameters: `dataset_path`, representing the path to the original training dataset, `output_path`, the path where the preprocessed images will be saved, and `target_size`, a tuple specifying the desired dimensions (height, width) for the images.

The function begins by creating the output directory (`output_path`) if it doesn't exist. It then utilizes the `ImageDataGenerator` from TensorFlow to perform image preprocessing, specifically rescaling pixel values to the range [0, 1]. The `flow_from_directory` method is employed to generate batches of preprocessed images from the specified training dataset directory. The parameters of this method include the target size, batch size, class mode (set to 'categorical' for multi-class classification), and additional settings for saving the preprocessed images.

The images are not shuffled during this process (`shuffle=False`), and the preprocessed images are saved to the specified output directory with filenames prefixed by 'preprocessed_' and saved in PNG format.

The code concludes by calling the `preprocess_and_save` function with specific paths for the original training dataset (`dataset_path`) and the desired output directory for preprocessed images (`output_path`). In this example, the target size for resizing the images is set to (224, 224).

3.2 Data Augmentation

The code performs data augmentation on a preprocessed flower image dataset and organizes the augmented images into a new directory structure. The functionality can be described as follows:

The code begins by defining paths for the original preprocessed dataset and the target augmented dataset. It then creates the output directory for the augmented dataset if it doesn't exist. Using the TensorFlow `ImageDataGenerator`, the code applies various augmentation techniques such as rotation, shifting, shearing, zooming, horizontal flipping, and brightness adjustment to each image in the original dataset. For each class in the original dataset, the code generates additional augmented images based on the specified augmentation parameters. The number of augmented images per original image is determined to achieve a target total count of 1000 images per class. The augmented images are saved in the new directory structure, with filenames indicating their augmentation status. The resulting augmented dataset is organized into subfolders corresponding to the original class structure. The code concludes by printing a message indicating the completion of data augmentation and organization.

4 Data Annotation

The code defines a class named `AutomaticAnnotationTool` designed to automate the annotation of images within a specified root folder. The class utilizes keypoint annotations and saves the annotations in a JSON file. The `AutomaticAnnotationTool` class is initialized with a root folder path and an annotation file path. Upon initialization, it checks if the annotation file already exists. If it does, it loads the existing annotations into the class instance. The main functionality is performed by the `annotate_images` method, which iterates through subfolders within the specified root folder. For each subfolder, it calls the `annotate_images_in_folder` method to annotate individual images. In the annotation process, keypoints are generated for each image using the `generate_keypoints` method, and the annotations are stored in a dictionary with image paths as keys. The annotations are then saved to the annotation file using the `save_annotations` method. Additionally, the class provides a method named `draw_annotations` to visualize the annotated keypoints on an image. In the example usage block, an instance of the class is created with specific root folder and annotation file paths, and the `annotate_images` method is called to execute the annotation process.

5 Data Localization

The code script encompasses a series of functions for refining and saving bounding boxes based on a pre-trained flower localization model. Additionally, it creates a new localized dataset by extracting regions of interest from flower images using these refined bounding boxes. Following is the detailed description of its functionality:

1. The code initially defines functions for loading annotations from a JSON file, loading image data along with keypoints, creating a convolutional neural network (CNN) model for flower localization, and training the localization model.
2. It then loads a pre-trained localization model and performs inference on a sample image, drawing the predicted bounding box for visualization.
3. Subsequently, the script introduces a refinement process for bounding boxes by isolating the flower region within each image using color-based segmentation.
4. The refined bounding boxes are then applied to the entire dataset.
5. The resulting refined annotations are saved in a new JSON file.
6. A localized dataset is generated by cropping the flower regions based on the refined bounding boxes.
7. Both the refined annotations and the localized dataset are stored in specified directories.

In summary, this script streamlines the process of refining bounding boxes for flower images, enhances the accuracy of localization through color-based segmentation, and creates a new localized dataset for further analysis or model training. The detailed steps include loading annotations, refining bounding boxes, saving refined annotations, and generating a localized dataset.

6 EfficientNet Overview

EfficientNet is a family of convolutional neural network (CNN) architectures that have gained popularity for their impressive performance and efficiency in terms of parameter size and computational cost. The architecture is designed to achieve state-of-the-art accuracy on image classification tasks while using fewer resources compared to traditional models. The EfficientNet model introduced a compound scaling method that simultaneously scales the depth, width, and resolution of the network to balance these factors for optimal performance.

6.1 Compound Scaling

EfficientNet introduces a novel compound scaling method to balance three critical dimensions of a CNN: depth, width, and resolution. The idea is to scale these dimensions uniformly to maintain a good balance between model complexity and computational efficiency.

6.2 Base Architecture

The base architecture of EfficientNet, denoted as B0, serves as the foundation. It consists of a stack of convolutional layers with a lightweight inverted bottleneck block as its basic building block. The inverted bottleneck block includes depthwise separable convolutions and linear bottlenecks, which contribute to the model's efficiency.

6.3 Scaling Factors

EfficientNet uses scaling coefficients (phi values) to uniformly scale the network's depth, width, and resolution. These coefficients are hyperparameters that determine the size of the network. The compound scaling formula is expressed as follows:

$$\begin{aligned}\text{Depth: } & d = \alpha^{\phi} \\ \text{Width: } & w = \beta^{\phi} \\ \text{Resolution: } & r = \gamma^{\phi}\end{aligned}$$

where α, β, γ are constants, and ϕ is the user-defined compound scaling parameter.

6.4 Efficient Blocks

The EfficientNet model utilizes a combination of different types of efficient building blocks, including inverted residuals with linear bottlenecks, depthwise separable convolutions, and efficient channel attention. These blocks contribute to both accuracy and computational efficiency.

6.5 Depthwise Separable Convolutions

Depthwise separable convolutions are a key component of the EfficientNet architecture. They factorize the standard convolution operation into depthwise convolutions and pointwise convolutions, reducing the number of parameters and computations.

6.6 Global Average Pooling (GAP)

The architecture employs global average pooling as a downsampling method, which reduces spatial dimensions and produces a compact global representation of the input.

6.7 Transfer Learning and Pre-training

EfficientNet is often used as a feature extractor for transfer learning. The base model, pre-trained on large-scale datasets like ImageNet, captures generic features that can be fine-tuned for specific tasks with smaller datasets.

6.8 EfficientNetB0 Configuration

In the code, the base model is EfficientNetB0 with an input shape of (256, 256, 3). The `include_top` parameter is set to `False` to exclude the final fully connected layer, and the model is initialized with weights pre-trained on ImageNet.

In summary, the EfficientNet architecture combines compound scaling, efficient building blocks, and depthwise separable convolutions to achieve state-of-the-art performance in image classification tasks with a high degree of efficiency in terms of computational resources. The code leverages the EfficientNetB0 model for a flower classification task, demonstrating the transfer learning approach and incorporating data augmentation during training.

7 ResNet Overview

7.1 Data Preparation

- The code loads images from training and testing directories (`train_data_path` and `test_data_path`).
- A subset of images (specified by `subset_size`) is loaded for each class.
- Images are resized to (128, 128) pixels.

7.2 Model Architecture

- The base of the model is ResNet50, a pre-trained deep convolutional neural network.
- The last classification layer of ResNet50 is removed (`include_top=False`), and a custom classifier is added on top.
- All layers of the base model are set to non-trainable (`layer.trainable = False`) to retain pre-trained weights.
- The custom classifier consists of a global average pooling layer, a dense layer with 1024 units and ReLU activation, a dropout layer with a dropout rate of 0.5, and a dense layer with softmax activation for classification.

7.3 Model Compilation

- The model is compiled using the Adam optimizer with a learning rate of 0.001, categorical crossentropy as the loss function, and accuracy as the evaluation metric.

7.4 Data Augmentation

- The training data is augmented on-the-fly using `ImageDataGenerator` with rescaling.

7.5 Training

- The model is trained on the augmented data for 100 epochs, with a validation set taken from 10% of the training data (`validation_steps=val_steps`).
- The training history is stored in the `history` variable.

7.6 Model Saving

- The trained model is saved to a file named 'RETRAIN_ResNet.h5' using `model.save()`.

7.7 Plotting Functions

- Two utility functions are defined for visualizing the model's performance:
  - `plot_roc_curve`: Plots the Receiver Operating Characteristic (ROC) curve for multi-class classification.
  - `plot_confusion_matrix`: Plots the confusion matrix.

7.8 Summary of the Model Architecture

- **Base Model:** ResNet50 (pre-trained on ImageNet).
- **Custom Classifier:**

  [label=–,left=0pt]Global Average Pooling Dense Layer with 1024 units and ReLU activation Dropout Layer (Dropout rate: 0.5) Dense Layer with Softmax activation (Output layer)
- This architecture leverages the pre-trained features of ResNet50 and fine-tunes the model on the flower dataset. The global average pooling layer helps reduce the spatial dimensions, and the dense layers add task-specific information for flower classification.
- The model uses the categorical crossentropy loss function, indicating a multi-class classification task. The code uses the `ImageDataGenerator` for on-the-fly data augmentation during training. Additionally, the learning rate, number of epochs, and other hyperparameters can be adjusted based on the specific requirements of the task.

8 Rationale of Design & Implementation

The design of the flower classification project is grounded in a well-considered rationale aimed at addressing the complexities inherent in recognizing and localizing diverse flower species. The following key rationales underpin the project's design:

1. **Dataset Curation and Augmentation:**
   - *Diversity for Generalization:* The project recognizes the importance of a diverse dataset that spans various flower types, capturing the intricacies of different species. Augmentation techniques are employed to enhance the dataset's variability, ensuring the model generalizes well across a wide spectrum of visual characteristics.
2. **Choice of Model Architecture:**
   - *ResNet for Feature Extraction:* The selection of the ResNet architecture is motivated by its efficacy in capturing intricate features in images. ResNet's deep residual learning helps mitigate the vanishing gradient problem, allowing the model to learn complex patterns crucial for accurate flower classification.
3. **Object Localization for Precision:**
   - *Bounding Box Refinement:* Object localization is recognized as a critical aspect of accurate flower classification. The design incorporates a specialized script to refine bounding box annotations, leveraging color-based segmentation for precise localization. This enhances the model's ability to focus on relevant regions within images.
4. **Comprehensive Evaluation Metrics:**
   - *Holistic Performance Analysis:* The choice of evaluation metrics, including ROC curves, precision-recall analysis, confusion matrices, and loss plots, ensures a comprehensive understanding of the model's performance. This multifaceted evaluation facilitates nuanced insights into classification accuracy, model convergence, and potential areas for improvement.
5. **Real-World Applicability:**
   - *Video Sequence Testing:* Recognizing the practical application of the model, a dedicated script is designed to process video sequences. This script not only visualizes the model's predictions but also creates annotated frames for qualitative assessment, offering insights into the model's performance in dynamic, real-world scenarios.
6. **Adaptability and Future Enhancement:**

- *Iterative Development:* The project design emphasizes an iterative approach, acknowledging the dynamic nature of machine learning. This allows for continuous monitoring, fine-tuning, and adaptability to evolving scenarios. Future enhancements could explore advanced architectures, transfer learning, and an expanded dataset to further improve model performance.

In essence, the rationale behind the project's design is rooted in the pursuit of developing a robust, accurate, and adaptable flower classification system. Each design choice is meticulously made to address specific challenges, ensuring a holistic approach to machine learning in the context of flower recognition and localization.

9 Evaluation

ROC Curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation of a binary classifier's performance across various threshold settings. It plots the true positive rate against the false positive rate, illustrating the trade-off between sensitivity and specificity. A higher area under the ROC curve indicates better classifier performance.

Precision & Recall

Precision and recall are metrics used to evaluate the performance of classification models, especially in imbalanced datasets. Precision is the ratio of true positive predictions to the total predicted positives, emphasizing the accuracy of positive predictions. Recall, or sensitivity, is the ratio of true positive predictions to the total actual positives, emphasizing the ability to capture all positive instances.

Confusion Matrix

A confusion matrix is a table that summarizes the performance of a classification algorithm. It displays the counts of true positive, true negative, false positive, and false negative predictions. From the confusion matrix, various metrics such as accuracy, precision, recall, and F1-score can be derived, providing a comprehensive view of model performance.

Loss Plots

Loss plots are graphical representations of the model's training and validation loss over epochs during the training process. Loss indicates how well the model is performing, and a decreasing loss suggests that the model is learning from the training data. Monitoring the loss plots helps identify issues such as overfitting or underfitting and guides adjustments in model architecture or training parameters.

10 Testing

10.1 Video Processing & Testing

The code functions as a post-processing and visualization tool for a flower classification model applied to a video sequence. The following outlines its detailed functionality:

- The script initiates by loading refined bounding box annotations, a pre-trained flower classification model, and a video file designated for testing.
- It establishes the output directory where frames, enriched with visual annotations, will be stored.
- While processing each frame from the video, the script preprocesses the frame by resizing it to the model's expected input shape and normalizing pixel values.
- The loaded model is then utilized to predict the flower class in the processed frame. Bounding boxes are overlaid around flower regions based on the refined annotations. Furthermore, the predicted flower class is presented as text within the bounding box.
- Each frame, accompanied by annotations and predictions, is saved as an image in the specified output directory.
- In a separate section, the script randomly selects and visualizes five frames from the output directory. It displays the original frame along with predicted flower class labels, offering a qualitative assessment of the model's performance on the chosen frames.

In summary, the script seamlessly integrates preprocessing, model inference, and visualization to showcase the flower classification model's predictions on a video sequence. The annotated frames, featuring bounding boxes and predicted class labels, provide valuable insights into the model's generalization across different flower instances in the video.

10.2 Image Testing

The script conducts testing on a set of flower images, utilizing a pre-trained ResNet model that was previously retrained on an augmented dataset. Here's a description of its functionality:

- The script begins by defining a function, `preprocess_frame`, which resizes an input frame to the desired shape (128, 128, 3) and normalizes pixel values to a floating-point scale between 0 and 1.
- It then sets the paths for the pre-trained ResNet model (`RETRAIN_ResNet.h5`) and the testing dataset containing flower images (`Test_Flower`).
- After loading the model, the script compiles it with the Adam optimizer and categorical crossentropy loss, which is appropriate for multi-class classification.

- The script defines the class names corresponding to different flower types.
- For testing, the script randomly selects 20 images from the testing dataset and preprocesses each image using the `preprocess_frame` function.
- It then uses the pre-trained ResNet model to predict the class of each processed image. The actual class is determined based on the directory structure of the image file.
- The script generates a visual output by creating a 4x5 grid of subplots, where each subplot displays an original flower image. The title of each subplot includes both the actual and predicted flower class.
- The resulting visualization provides an overview of how well the retrained ResNet model performs on the selected sample of flower images, highlighting any correct or incorrect predictions.

In summary, this script serves as a visual assessment tool, displaying the actual and predicted classes for a subset of flower images from the testing dataset. It offers insights into the model's performance on individual flower instances and helps identify potential areas for improvement in the classification task.

11 Intersection over Union (IoU)

The script performs testing on a set of 20 randomly selected flower images from a testing dataset (`Test_Flower`). It utilizes a pre-trained ResNet model (`RETRAIN_ResNet.h5`) for flower classification. The script begins by defining a preprocessing function, `preprocess_frame`, which resizes an input image to the desired shape (128, 128, 3) and normalizes pixel values to a floating-point scale between 0 and 1.

For each randomly selected image, the script loads and preprocesses it, then uses the pre-trained ResNet model to predict the flower class. The actual flower class is determined based on the directory structure of the image file. The script visualizes each original image along with the actual and predicted flower classes in a 4x5 grid of subplots.

Additionally, the script calculates the Intersection over Union (IoU) for each image. In this case, it uses a ground truth bounding box (`ground_truth_box`) with coordinates [0, 0, 50, 50]. The IoU is printed for each image, representing the overlap between the predicted and ground truth bounding boxes.

In summary, the script serves as a visual and quantitative assessment tool. It visually displays the actual and predicted classes for a subset of flower images, providing insights into the model's performance. Moreover, it calculates the IoU as a measure of the localization accuracy of the predicted bounding box compared to the ground truth, enhancing the evaluation of the model's object detection capabilities.

12 Potential Improvements

The flower classification project can benefit from several potential improvements to enhance its accuracy, robustness, and adaptability. Firstly, exploring advanced neural network architectures, such as EfficientNet, DenseNet, or NASNet, could provide improved feature extraction capabilities. Additionally, leveraging transfer learning techniques by fine-tuning a pre-trained model on the flower dataset may enhance the model's generalization. Enriching the dataset with a more extensive collection of flower images, increasing data augmentation variability, and fine-tuning hyperparameters are essential steps to improve overall performance. Dynamic bounding box refinement techniques that adapt to varying flower sizes and shapes could enhance localization accuracy. Ensemble learning, combining predictions from multiple models, and implementing continuous monitoring and updating systems for the model contribute to sustained effectiveness over time. These strategies collectively aim to elevate the flower classification project to achieve superior results and adaptability.

13 Conclusion

In conclusion, the flower classification project has been a comprehensive exploration into the realm of computer vision and machine learning. The primary objective was to develop an effective model for accurately classifying different types of flowers. Throughout the project lifecycle, several key components and processes were implemented, providing a holistic approach to achieving this goal.

The initial phases involved data preprocessing, encompassing the curation of a diverse and augmented dataset. This dataset was instrumental in training a ResNet model, which served as the core engine for flower classification. The training process was iterative, involving fine-tuning and optimization to enhance the model's ability to generalize across various flower types.

Moreover, the project delved into the crucial aspect of object localization. A specialized script was developed to refine bounding box annotations, improving the accuracy of flower localization within images. This script not only contributed to the model's precision but also facilitated the creation of a localized dataset, further enhancing the robustness of the system.

The evaluation phase was multifaceted, incorporating standard metrics such as ROC curves, precision-recall analysis, and confusion matrices. This facilitated a nuanced understanding of the model's performance across different dimensions, aiding in the identification of strengths and areas for improvement. Additionally, the use of loss plots provided insights into the training process and convergence of the model.

The testing phase demonstrated the model's application to real-world scenarios, particularly in the context of video sequences. The post-processing script efficiently visualized the model's predictions, creating annotated frames for qualitative assessment. The random selection and visualization of frames provided a snapshot of the model's performance on diverse flower instances.

The project's success is underlined by the synergy between the developed scripts, the curated dataset, and the trained ResNet model. The combination of these elements resulted in a flower classification system that not only predicts classes accurately but also localizes flowers within images effectively.

However, as with any project, there are avenues for future enhancements. These include exploring more advanced architectures, integrating transfer learning techniques, and expanding the dataset to encompass an even broader spectrum of flower variations. Furthermore, continuous monitoring and fine-tuning will be essential to adapt the model to evolving scenarios and ensure its longevity.

In essence, the flower classification project showcases the amalgamation of data science, machine learning, and computer vision techniques to tackle a real-world problem. The journey from data preprocessing to model deployment has provided valuable insights, underscoring the dynamic and iterative nature of developing robust machine learning systems.