Image classification with distribution shift

Manuela Lo Giudice MANUELA99LG@GMAIL.COM

1. Dataset

This Homework is a Kaggle competition with a dataset provided by the Professor.The dataset consists of images of handheld objects and exhibits a domain shift issue between the training and test data. The training set contains 1600 images, with 200 images for each of the 8 classes. The test set, on the other hand, contains 800 images. All the images have an original dimension of 350x350 pixels.

One notable characteristic of the dataset is that class 0, which represents plug adapters, has the same background in all the test set images. This results in a domain shift, indicating a change in the distribution of the data.

To address the challenge posed by domain shift and enhance performance, bounding boxes were provided for the image classification task. These bounding boxes are available in a .csv file and provide information such as image id, coordinates (x1, y1, x2, y2), and class label. By utilizing these bounding boxes, we can restrict the model's focus to the specific area defined by the coordinates, improving its ability to classify objects accurately.

Figure 1: Example of dataset per classes

2. Model Description

ResNet-18 is a specific variant of the ResNet (Residual Network) architecture, which was introduced by Microsoft Research in 2015. It is a convolutional neural network (CNN) model primarily used for computer vision tasks such as image classification, object detection, and segmentation. ResNet-18 is known for its relatively compact architecture while still achieving high accuracy.

Now, let's delve into a detailed description of the ResNet-18 architecture:

© 2023 M.L. Giudice.

- **"Input Layer"**: takes an input image with three color channels (RGB) and a fixed size of 224x224 pixels.

- **"Entry flow"**: The initial layer is a 7x7 convolutional layer with a stride of 2, followed by a 3x3 max pooling layer with a stride of 2. These layers reduce the spatial dimensions of the input image.

- **"Residual Blocks"**: ResNet-18 consists of four sets of residual blocks, each containing multiple convolutional layers. The first set has two residual blocks. Each block consists of two 3x3 convolutional layers with padding, followed by a skip connection that adds the input to the output of the block. The convolutional layers use a stride of 1 and produce 64 output channels. The second set also has two residual blocks. Each block follows the same structure as the blocks in the first set, but the number of output channels is increased to 128. The third set includes two residual blocks, similar to the previous sets, but with 256 output channels. The fourth set contains two residual blocks with 512 output channels. The skip connections in each residual block allow the network to learn residual mappings by capturing the difference between the input and output of a block. This helps alleviate the vanishing gradient problem and facilitates the training of deep networks.

- **"Global Average Pooling"**: After the residual blocks, a global average pooling layer is applied. It reduces the spatial dimensions of the feature maps to a fixed size, resulting in a 512-dimensional feature vector. Global average pooling computes the average value of each feature map across its spatial dimensions.

- **"Fully Connected Layer"**: The 512-dimensional feature vector is fed into a fully connected layer with a softmax activation function. The number of neurons in this layer corresponds to the number of classes in the classification task, and it produces the final probability distribution over the classes.

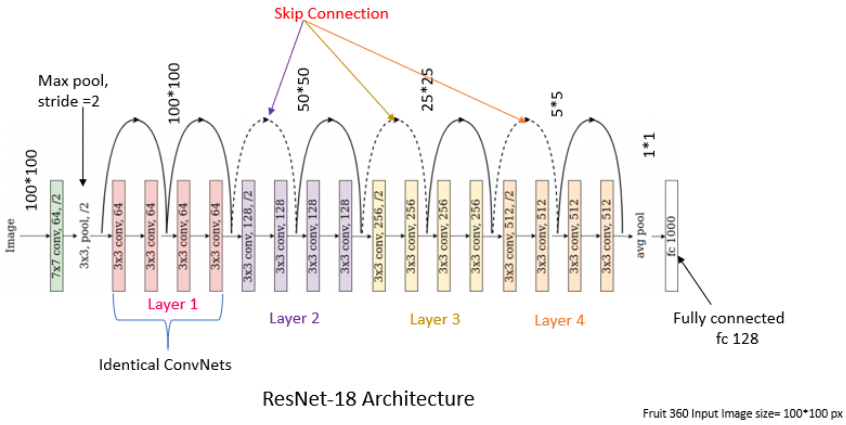

Figure 2: ResNet-18 Architecture

As a conclusion, ResNet-18 is composed of an initial convolutional layer, followed by four sets of residual blocks with varying numbers of output channels. It also includes global average pooling and a fully connected layer for classification. The introduction of skip connections in the residual blocks allows ResNet-18 to effectively address the challenges of training deep neural networks and achieve improved accuracy.

3. Training procedure

3.1. Image Segmentation

3.1.1. HSV Scale

The primary method employed to mitigate the distribution discrepancy between class "0" and the other classes is through Image Segmentation. Hence, the segmentation process is exclusively applied to images classified as belonging to class "0".

The initial step entails converting the images from the RGB color space to HSV, resulting in three-channel images where each channel represents a different aspect: Hue, Saturation, and Value.

The background was chosen by applying a threshold to the pixel values based on the range of Hue [36, 89], Saturation [25, 255], and Value [25, 255]. A binary mask was then created to exclude the selected background from the final image.

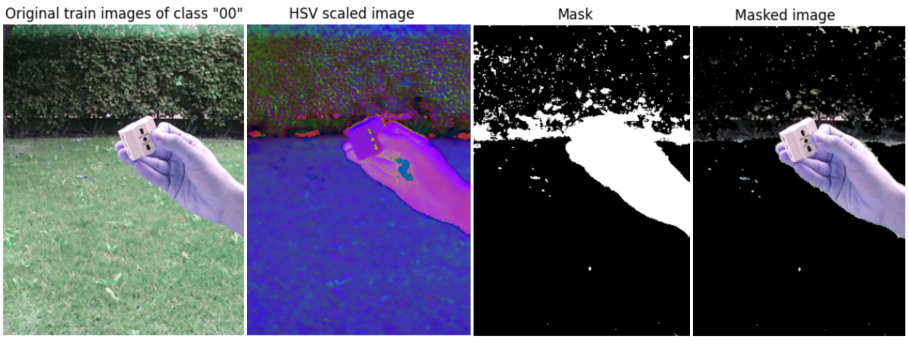

Figure 3: Background Selection and Binary Mask Construction

This step effectively eliminated the initial problematic details from the class "0" images.

3.2. Image Crop

The purpose of this second method is to automate the process of cropping images based on predefined bounding box coordinates. I take the directory of images, along with the CSV file containing the bounding box coordinates for each image, and crop the images accordingly. The cropped images are then saved in a separate output

directory. I applied this method for both train and test data to improve model performance and make it more efficient and convenient. The code serves multiple purposes, primarily focused on facilitating data preparation for computer vision tasks. The process of cropping images based on bounding box coordinates allows for the extraction of specific regions of interest within the images. This is particularly valuable for tasks such as object detection, where the goal is to identify and locate objects within an image accurately. By isolating the relevant regions, the code assists in creating quality training and testing datasets and improving the overall performance of computer vision models.

Figure 4: Example of images cropped

3.3. Data Transformer

The second important approach was the data transformer.

The train transformation:

- Resizes the image to a fixed size of 224x224 pixels. This is a common practice to ensure consistent input dimensions for deep learning models,
- Randomly flips the image horizontally with probability of 0.5. This augmentation technique helps introduce variations and increase the robustness of the model during training,
- Randomly rotates the image by a certain degree (in this case, 10 degrees) in a counter-clockwise direction. This introduces further variations and helps the model learn to recognize objects from different angles,
- Converts the image from PIL Image format to a PyTorch tensor. This is necessary for further processing and feeding the image into a deep learning model,
- Normalizes the image tensor by subtracting the mean values and dividing by the standard deviation values. This normalization step is crucial for improving the convergence and stability of the training process. The mean and standard de-

viation values provided ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) are standard values often used with pre-trained models trained.

The test transformation:

- Resizes the image to a fixed size of 224x224 pixels, similar to the train transformation,
- Converts the image from PIL Image format to a PyTorch tensor,
- Finally, normalizes the image tensor using the same mean and standard deviation values as in the train transformation.

3.4. Hyperparameters

The hyperparameters used for train a ResNet18 model are Adam optimizer with a learning rate of 0.0003 and weight decay of 0.0001. I also set the loss function to CrossEntropyLoss.

Let's analyze it in depth:

- Adam optimizer: The optimizer is responsible for updating the model's parameters during the training process. In this case, I used the Adam optimizer, which is a popular optimization algorithm known for its adaptive learning rate. The learning rate for the optimizer is set to 3e-4 (0.0003), which determines the step size for parameter updates. The weight decay parameter of 0.0001 is used for L2 regularization to prevent overfitting.
- CrossEntropyLoss: The loss function computes the discrepancy between the predicted outputs and the true labels, providing a measure of how well the model is performing. Cross-entropy loss is commonly used for multi-class classification tasks, so right in our case.

Optimizer	Adam optimizer
Learning Rate	3e-4 (0.0003)
Regularization	Weight Decay 0.0001

4. Experimental Results

The model has been trained for 10 epochs and these are the following training loss for each epoch computed:

N° epochs	Training Loss
0	0.2611
1	0.0177
2	0.0049
3	0.4349
4	0.0083
5	0.4194
6	0.1216
7	0.0091
8	0.0019
9	0.0445

Table 1: Training loss for each epoch computed

During the training process, the model's performance can be evaluated by examining the training loss values for each epoch. In this case, the initial epoch started with a loss value of 0.2611, which decreased significantly to 0.0177 in the second epoch. As training continued, the loss consistently decreased, reaching a minimum of 0.0019 in the eighth epoch.

However, it is important to highlight two notable fluctuations in the training loss. The third epoch exhibited an increase to 0.4349, while the fifth epoch showed a rise to 0.4194. These fluctuations indicate that the model's performance temporarily regressed during those particular epochs. Possible factors contributing to these fluctuations include noise in the training data or an insufficient amount of data.

Despite these fluctuations, the overall trend of decreasing training loss implies that the model gradually improved its performance over time. It signifies that the model effectively learned from the training data and adjusted its parameters to minimize the loss. This decrease in loss values suggests that the model gained a better understanding of the underlying patterns in the data, leading to improved performance in subsequent epochs.

In the given example, the model demonstrated accurate recognition of the sunglasses, the light bulb, and the can. However, it encountered difficulties in identifying the socket and mistakenly classified it as a telephone.

Figure 5: Output Example of Resnet-18

In the end, by leveraging the power of a pre-trained Resnet18 model, I achieved an impressive accuracy of **89.5%**. This model, which was specifically trained for the task at hand, has the capability to capture personalized features, resulting in highly satisfactory and realistic outcomes.

4.1. Captum: Guided Grad-CAM

Captum is a Python library that allows us to understand the behavior and decisions made by these models. One of the techniques supported by Captum is Guided Grad-CAM. Grad-CAM stands for Gradient-weighted Class Activation Mapping. It is a technique used for visualizing and understanding the important regions in an input image that contribute the most to the predictions made by a deep-learning model. Grad-CAM generates a heat map that highlights the regions of the image that are important for the model's decision. In this case, it was able to generate and visualize attribution maps in the pre-trained ResNet-18 model.

Figure 6: Grad-CAM Output Example of Resnet-18