Deep Learning Assignment 7 in FS 2022

Transfer Learning

Microsoft Forms Document:
https://forms.office.com/r/MvPiCwh6jR

Manuel Günther

Distributed: Friday, April 8, 2022
Discussed: Friday, April 29, 2022

The goal of this exercise is to learn how to use pre-trained networks in transfer learning tasks. We will make use of networks trained on ImageNet, and apply them to related problems, i.e., the classification of fruits and vegetables.

For this exercise we use parts of the Fruits and Vegetables dataset that can be downloaded from Kaggle.[1] Since the dataset is quite large, I have subselected some images, which are available on Seafile.[2] In each directory, which represents a class, I have stored two images, `gallery.jpg` used for training and `probe.jpg` for testing. You might want to have a look into these files to see the directory structure. This dataset will automatically be downloaded within our Jupyter Notebook template.

1 Dataset

Since we will use a network pretrained on ImageNet, we will need to perform the exact same image transformation steps as used by PyTorch to train the networks. While detailed instructions can be found on the PyTorch documentation,[3] we can boil them down to 4 steps:

1. The image is resized such that the shorter side has size 256.
2. The center crop of size 224×224 is taken.
3. The image is converted to a `torch.Tensor` and its values are scaled to be in range $[0, 1]$.
4. The image is normalized using mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$.

While we could implement the dataset loading on our own, here we rely on PyTorch functionality. Particularly, we make use of the `torchvision.datasets.ImageFolder` class, which can be equipped with a `transform` instance.

Task 1: Data Transformation Instantiate a transform according to the above instructions.

Task 2: Dataset Loading Create two datasets, one for training and one for testing. You might want to make use of the `is_valid_file` parameter of `torchvision.datasets.ImageFolder` to select the files for the two datasets.

Test 1: Data Sizes and Types Check that the datasets are as expected. Check that both contain the same number of classes, and that each dataset has exactly one image per class. Check that all images are of size $3 \times 224 \times 224$, and that their `dtype` is `torch.float`.

[1] https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition
[2] https://seafile.ifi.uzh.ch/f/72e1d9c4ef20420eb1d9/?dl=1
[3] https://pytorch.org/vision/stable/models.html

2 Deep Feature Extraction

We will use a pretrained network available from PyTorch.[3] Particularly, we will use a RseNet-18 architecture, but other architectures can also be tested. Fortunately, PyTorch provides simple interfaces to obtain pre-trained models, e.g., using the `torchvision.models.resnet18` interface function.

In order to use the networks in a transfer learning task, we need to change their outputs. There are several possibilities on how to achieve that, and you have the freedom to choose. For your reference, the original implementation of the `forward` function can be found on GitHub.[4]

Task 3: Pre-trained Network Instantiate a pre-trained ResNet-18 network. Make sure to extract the deep feature layer, i.e., implement a way to remove the last `fc` layer from the network.

Task 4: Extract Features Implement a function that extracts the feature for a given dataset and a given network. Remember that the network requires a 4-dimensional tensor, i.e., it works on batches. Return the features in a dictionary with the class index as key and the extracted features as values. Call this function for both the training and test set.

Test 2: Check your Features Check that all of your features are of dimension 512 (for smaller networks like ResNet-18) or 2048 for larger networks. Check that the datatype is `torch.float`.

[4] https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py#L264

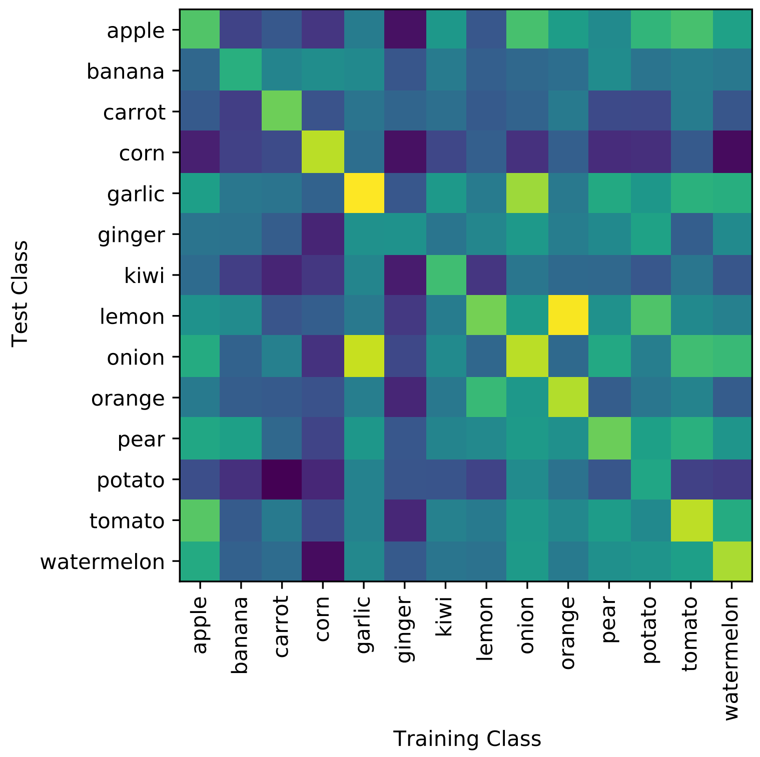

Figure 1: This figure shows an exemplary similarity matrix of the different classes.

3 Evaluation

For assigning a class to a test sample, we need to compare the test feature with all the training features. Finally, we select the class of the training feature with the highest similarity to the test set.

Task 5: Similarity Computation Compute the similarities between all test features and all training features and store all of them in a matrix. Here, we use the cosine similarity function:

$$\cos(\vec{a}, \vec{b}) = \frac{\vec{a}^{\mathrm{T}}\vec{b}}{\|\vec{a}\|\|\vec{b}\|} \tag{1}$$

Store the similarities in a matrix of size $O \times O$, where the first index is the true target class and the second index is the predicted class.

Task 6: Plot Similarity Values Use `matplotlib` functionality to plot the matrix of similarities as an image, see Fig. 1 for an example.

Task 7: Classification Accuracy A test sample is correctly classified if the highest similarity value is assigned to the correct class. Using the computed matrix of similarity values, compute the classification accuracy that we can obtain on the dataset.

Task 8: Misclassified Images Find the test samples that are misclassified. For each, provide the class names (the name of the fruit/vegetable, i.e., the directory name in our dataset) for the test sample and its prediction. Also provide the names of the two most dissimilar classes.