Programming Task 2

Task general description

In this task you will

* Build a flower classifier with task transfer from a pre-trained network (ResNet50V2).
* Try to Improve the classifier using at least two of several options: data augmentation in training, weight decay, architecture changes, using another network or other ideas of your own.
* Test your classifier’s accuracy and report your results

The exercise goals: experience building a classification pipe with CNN components. Get acquainted with some possible methods for pipe improvement.

Dataset

The flowers dataset was collected in Volkani institute:

The data includes 472 cropped images of flowers and non-flowers, with corresponding labels. The original images from which the flower and non-flower rectangles were cropped from are not given, due to their large size. The data is in 'FlowerData.zip'. It includes the images and also the labels in a file named 'FlowersDataLabels.mat'. Use the command scipy.io.loadmat() to read it.

You will have to resize the images to 224\*224\*3, which is the input shape of the pre-trained CNNs on ImageNet. Notice that in some network implementations the input shape might be 227\*227\*3.

In your implementation, in the beginning of the “main” section, define a variable named *test\_images\_indices* which will hold a list containing the numbers of the images to be used as a test set. The remaining images will be used as the training set. For example:

test\_images\_indices = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

In this case, the first 40 images in the FlowerData folder (files named 1-40 + “.jpeg”) will be used for the test set. The next images, named 41-472, will be used as the training set.

For your own work, submitted code and report, use the first 300 images for training, and test on the remaining 172 (i.e., test\_images\_indices = list(range(301,473)) ). When I test your work, I will change the test set images using the test\_images\_indices variable.

Task description

The Basic pipe includes:

* Initiate an [ResNet50V2](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf)-based architecture (using the [Keras](https://keras.io/) public library):
  + [ResNet50V2](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf) implementation was built to classify images on the ImageNet dataset, which has 1000 classes. Hence the output layer has 1000 neurons.

In this project, the classification is flower or not flower. Thus, the output layer should have a single neuron estimating (i.e. the probability that an image is a flower). We will not use the fully-connected 1000-neurons layer at the top of the [ResNet50V2](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf) network. Instead, the layer before last , which is a should be globally pooled (average pooling) over the spatial dimensions to get a vector of elements, with . Then, the output neuron for our net will be obtained as with .

See the Keras ResNet documentation for more details: <https://keras.io/applications/#resnet>

* + The loss: Assume an example with the label ( means that the image is a flower, means it is not), and the network output is . We minimize

For a single image. The total loss is .

This network loss in Keras is called "binary\_crossentropy".

* + **Important remark regarding Keras** – Keras runs on top of other deep learning frameworks. In this work, use [Tensorflow](https://github.com/tensorflow/tensorflow) as Keras backend.
* Task transfer: use pre-trained [ResNet50V2](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf) weights (trained on ImageNet dataset) as the initial weights of the network. They can be loaded to the network when executing the command for loading the network (keras.applications.resnet\_v2.ResNet50V2… see the Keras ResNet documentation for more details).
* Fine tune the network weights for the specific problem. Tuning all the network weights is not necessarily the best idea (error-wise and time-wise) - it is recommended to start by trying to tune the final layers only.

In addition to implementing the basic CNN pipe, you should try at least two ways to improve its results. Options are:

* Data Augmentation: Add to the training data (but not to the test data) examples created by applying horizontal flip and/or mild cropping of the original training images (and/or other distortions). A 3D image can be horizontally flipped by taking I2=I[:, ::-1, :]. When cropping, use random crops of large portions (>80-90%) of the original image. More explanations and examples can be found [here](https://www.cis.rit.edu/class/simg782/lectures/lecture_02/lec782_05_02.pdf).
* Weight decay (and tuning of it)
* Changing the architecture - some examples:
  + Pooling type: you may try max pooling instead of average pooling on the before-last layer, or different pooling parameters (local pooling with some stride and cell size instead of global pooling). You may also avoid pooling completely and use all the neurons by using flattening: turning the tensor into a vector of dimension 7\*7\*2048=100,352 dimensions, which then feeds the final neuron (see <https://keras.io/layers/core/> for flatten documentation)
  + Add another global layer before the final neuron
  + …
* Dropout (and tuning of it): a very commonly used regularization technique. Additional information [here](http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf) (see class 9 appendix for a 1-slide explanation)
* Neural decay (and tuning of it. see class 9 appendix for a 1-slide explanation)
* Using other networks (not ResNet50) as backbone
* You are welcome to propose other ideas and report their results.

Submission

You will submit the code and a short report of the results. (a single .py file, and a single .doc/.pdf file)

Code

* The code is required to run directly, without modifications, on my machine.
* The only expected modification in the code will be of path to the FlowerData dataset folder - it should appear at the beginning of the “main” code in a variable named *data\_path*. The *test\_images\_indices* variable will be defined with *list(range(301,473))* (I will modify it later in my testing).
* This machine will have python 3.7.5 and packages with the following versions: tensorflow 2.0.0, keras 2.3.1, numpy 1.18.1, scipy 1.4.1, cv2 4.1.2, scikit-learn 0.22.1. Please work with an environment using these python and package versions. You can also use other packages available in “pip” - mention it in your report and at the beginning of your code (including package version).
* The code is expected to be with similar structure to what we have discussed in the previous assignment, i.e.: it should include stages of preparing experiment parametrization, getting the data, representing it (In the CNN case representation usually means only resizing the image to the right size), train, test, evaluate and report. It is expected to be well documented!
* The code you submit should run the best model you have found and configured, which must run in less than ~25 minutes on your CPU machine (so I can check it on my computer in reasonable time).
* The code should use fixed hyper parameters (the best configuration of hyper-parameters you found).
* The code should print to the console the pipeline stage it is currently running. For example: “Importing data”, “training network”, “testing” etc.
* At the end of the run a “report” will be printed to the console, which includes: the number of training images and the amount of “True” flower images out of them (if augmented training set, the report will include this information for original images and augmented images separately), precision-recall curve AUC score, test error rate, confusion matrix (for a threshold of your choice), final model structure (using the keras model.summary() command), time in minutes of entire run.
* The code will also output a figure with Precision-recall curve obtained (to the user screen, using plt.show() ).
* At the beginning of the code (after imports), set numpy, random and tensorflow packages seed to 10, using commands: random.seed(10), np.random.seed(10), tf.random.set\_seed(10).
* Important: the code **will not** write or change files in the operating system folders!!! (including the data folder)

Report

The report should include

* Description of the hyper parameters values used for the main parameters (learning rate, number of epochs, batch size, weight decay (if used), early stopping criterion (if used)) and how they were chosen. If any other hyper parameters were tuned for your improved model, describe how you chose them as well. Don’t forget to use a validation set for parameter tuning and to describe the validation procedure you chose (how the training set was split into ‘training’ and ‘validation’ sets, or cross validation subsets, and how many images in each).
* Description of the improvement techniques you applied
* The test error rate and confusion matrix obtained on the test set, using the basic ResNet50V2 and also using the improvement techniques you applied.
  + Report the results of the basic pipe (before improvements) and the results of the best pipe configuration (after improvements) you have found.
* Error images:

The 5 worst errors of type 1 and 5 worst errors of type 2 (on the test set)

* + Type 1: false alarm: the algorithm thought it is an object, but it is not
  + Type 2: miss-detection: the algorithm thought it is not an object, but it is

For each error, print the error type (1 or 2), error index (1-5) and the classification score (the CNN score) in its title. The error images should be included in the written report, but should not be outputted by the submitted code.

* A recall-precision curve:
  + The CNN classifies according to the output neuron estimating . If this value is greater than a threshold *t = 0.5*, the image is classified as a flower, else it is classified as non-flower. However, we can change the threshold to be a different value. If we decrease *t,* more examples will be classified as flowers (and also getting more type 2 errors) and vice versa. Define:
    - Recall: – the fraction of correctly detected objects
    - Precision – the fraction of objects among the algorithm’s positive outputs (i.e. the examples the algorithm claims to be objects)

( is the number of examples the algorithm claims to be objects)

By varying between 0 and 1 we can obtain a recall-precision curve composed of achievable (recall, precision) points. Plot the full recall-precision curve with recall on the x-axis, precision on the y-axis. Also, calculate the area under this curve and write it in the report ( sklearn.metrics.auc() can help you do that).

Additional information

* Keras further information:
  + “Keras is a high-level neural networks API that allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).”
  + We choose to work with Keras on top of **Tensorflow**. TensorFlow is an open source software library for numerical computation using data flow graphs.

The installation of both Keras and TensorFlow are quite simple and can be found here:

[https://keras.io/#installation](https://keras.io/)

<https://github.com/tensorflow/tensorflow>

They are both available using pip.

* Example of transfer learning with a Keras pretrained network: <https://www.tensorflow.org/tutorials/images/transfer_learning>
* It is recommended to get yourself acquainted with the Keras sequential model (<https://keras.io/getting-started/sequential-model-guide/>) and also its functional API (<https://keras.io/getting-started/functional-api-guide/>)
* For ResNet50V2 on keras see <https://keras.io/api/applications/resnet/>
* Also nice to get familiar with - TensorBoard, a data visualization toolkit for deep NN experiments:

<https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard>