LabMLISP: Lab Course Machine Learning in Signal Processing SS 2019

Part VI: Data augmentations

Prof. Veniamin Morgenshtern

Author: Christoph Hertle

In this part of the lab course you will implement and apply data augmentation. In training neural networks a sufficiently large and divers training data set is critical. Acquiring data is usually an expensive and time consuming task. Therefore one might consider enhancing the size and diversity of a data set by implementing and applying data augmentation in order to mitigate this problem. This is what we will do in the following.

In the folder utils we already provided a starter code in the file augmentation.py. Your task is to implement data augmentation by completing the code in augmentation.py. We will use inter alia the following Python packages in order to implement the augmentation functions: random, numpy, PIL. We already imported these packages for you so you do not have to worry about that.

One way to augment data in the form of pictures is to apply geometric transformations to the images. These augmentation functions are easy to implement but can prove to be quite effective nonetheless.

Implementing Data Augmentation

Mirror Images You will now implement an augmentation function that performs data augmentation by mirroring images. You will do this by inserting the code inside the if-condition of the definition of the __call__ method of the LeftRightFlip class.

Hints: The __call__ function has apart from self the arguments image and label. image and label are both PIL Image objects (class PIL.Image.Image). The PIL module ImageOps provides functions to transform PIL Image objects. You can easily find out the name of the appropriate ImageOps function and how to use it by a quick web search[1]. Mirror both the image and the label.

Rotate Images You will now implement an augmentation function that performs data augmentation by mirroring images. You will do this by inserting the code inside the if-condition of the definition of the __call__ method of the Rotate class.

Hints:

- Again use a built-in function of the ImageOps module to implement the augmentation function.
- Rotate the images only within a reasonable range of angles (e.g. -70° to +70°). This range is defined in the file config.json ("max_rot") and in Augmentation object stored in the variable self.params['max_rot'].

[1] The documentation of the ImageOps module `https://pillow.readthedocs.io/en/3.0.x/reference/ImageOps.html` might especially be helpful.

- From this range choose the rotation angle randomly from a uniform distribution.
- Apply the rotation to both the image and the label.

Add Gaussian Noise to Images Another possibility to improve the performance of a neural network is to add Gaussian noise to the images. You will do this by inserting the code inside the if-condition of the definition of the __call__ method of the GaussianNoise class.

Hints:

- First you might want to convert the PIL Image object into a numpy array. Use the function numpy.array to do so. This will give you a (x_pixels x y_pixels x 3)-dimensional numpy array. Where x_pixels is the number of pixels in x direction of the image, y_pixels is the number of pixels in y direction of the image and each pixel has three color channels. To add the Gaussian noise you might want to use float numbers. However, take care that before you transform the numpy array back into a PIL Image object to convert the numpy array into uint8-numbers in a reasonable manner.
- Use PIL.Image.fromarray to convert the numpy array that now is in uint8 format back into a PIL Image object
- Add Gaussian Noise only to the image.

Testing Data Augmentation

Copy the notebook to the working directory:

```
$ cd ~/labmlisp

$ cp -r ~/SHARED/DATA/mlisp-lab/ps6_augmentation/* .
```

Use the AugmentationTest.ipynb notebook to test the augmentation functions you implemented above. Concretely, go to the part where you find augmentation_operation=Compose([Rotate(cfg_path)],cfg_path). Instead of Rotate you can use any other augmentation defined in augmentation.py. Or a combination of augmentations e.g. augmentation_operation=Compose([GaussianNoise(cfg_path), Rotate(cfg_path)],cfg_path). Note that the order of the augmentation matters. E.g. it does not make much sense to apply Rotate before GaussianNoise.

Use Data Augmentation to Train the Neural Network

In order to train the Neural Network use the Train.ipynb notebook. Again use e.g. augmentation_operation = Compose([LeftRightFlip(cfg_path), Rotate(cfg_path)], cfg_path) to define the augmentations to be applied. You augment the data in the SimulationDataset by SimulationDataset(new_dataset=True, size=100, augmentation=augmentation_operation) for example. If you don't pass the argument augmentation=augmentation_operation no augmentation would be applied

to the SimulationDataset. Similarly you can apply data augmentation to the TrueNegativDataset or not.

Experimentation

Now you are allowed to experiment a bit, i.e., train the network with different data augmentations and combinations of data augmentations[2]. Try as well different parameters for the various augmentations. Save the trained network with meaningful names so that you can remember with which augmentations the network was trained. Which augmentations prove to be most effective, i.e., which augmentation increase the number of true positive pixels and decrease the number of false positive pixels best?

[2]For sure you can also use the augmentations that were already provided completely in the starter code.