

BS6207 Assignment4 Report

Xinxin | Apr/2021

The assignment was tackled by the following 3 steps:

Step 1. Sort and Label the images

The images were provided in 4 separate folders based on the class label. To prepare the images for neural network, the following steps are taken:

1. Modify the image name to include class labels. For example, all images in the `artifacts` folder will be renamed as `artifactsXXX.png`, where XXX is the running number.
2. Take the first 100 images from each class as the test set. Copy and paste these 400 images together into a separate folder (`test_data`)
3. Put the rest of the images into a separate folder by class (e.g. `artifacts_train_val`), which contains the images to be used for both training and validation later)
4. Check the number of images in each class folder for training and validation:
 - artifacts: 2280
 - cancer_region: 2778
 - normal_region: 1087
 - others: 384
5. There is a class imbalance problem that `others` class is particularly small. Hence, image augmentation is applied to `normal_region` and `others` classes to generate more images.
 - For each `normal_region` image, 1 augmented image is generated. For each `others` class image, 6 augmented images are generated.
 - The augmented image may be generated by flipping, cropping, scaling and rotating the original image:
 - `Fliplr = 0.5` -> 50% of the images will be flipped horizontally
 - `Crop(percent = (0,0.1))` -> crop some of the images by 0-10% of their height/width
 - `Affine(scale = {'x': (0.9,1.3), 'y': (0.9,1.3)}, rotate=(-25,25))` -> scale to 90% -130% of image height or width; rotate by -25 to +25 degree
6. Check the number of images per class after augmentation:
 - artifacts: 2280
 - cancer_region: 2778
 - normal_region: 2174
 - others: 2688

The classes are roughly balanced.

Step 2. Build a CNN network and fine-tune the parameters

To build the CNN network, the following steps are taken:

1. From each train_validation folder, randomly select 70% of the images as training, and the rest will be used for validation.
2. Based on the image file names, generate the training/validation label list, which contains string labels (e.g. normal, cancer, etc.)

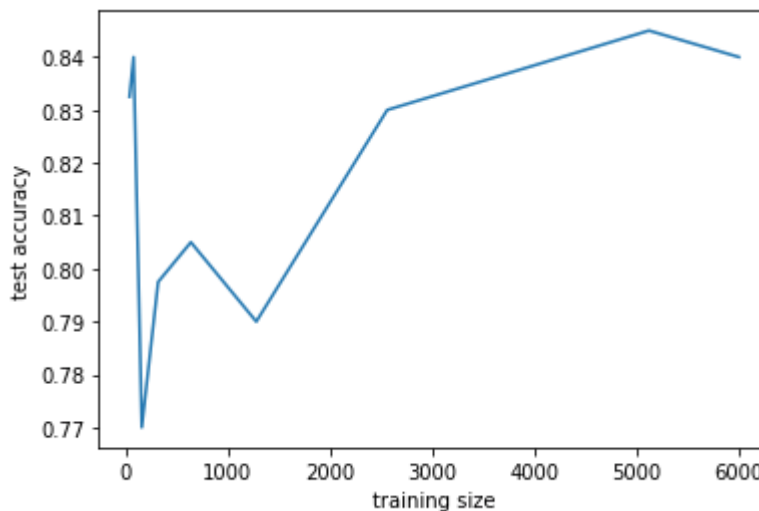
3. Read the images based on the train/validation filename list, and then convert the images into arrays. Each image is converted to a (1,128,128) array, where the images were imported as gray scale (i.e. one channel only).
4. The arrays are then normalized by dividing all pixel values by 255.
5. Convert the label list from string to numerical levels (i.e. 0~3) using `sklearn.LabelEncoder`.
6. Define the training and validation dataloader, where the trainloader has a batch size of 50, and valloader has a batch size of 1.
7. Build the test data loader using the same approach.
8. Build the CNN model:
 - The CNN model contains 3 CNN layers. After each CNN layers, the output is processed by:
 - ReLU - to introduce non-linearity in the image
 - batch normalization - standardize the inputs to a layer for each mini-batch
 - Maxpooling2D - generates pooled features maps that highlight the most present features from the image
 - After the 3 CNN layers, 2 densely connected layers are introduced with dropout and ReLU.
 - The output of the CNN model is raw score, for the convenience of using CrossEntropy as the loss function
9. Techniques used to fine-tune the CNN models include:
 - Dropout - 50% of the nodes are dropped out randomly to prevent overfitting
 - Adam gradient descent is employed to find the minima, which is better than stochastic gradient descent and is less likely to stuck at local minima
 - Weight decay of 0.005 is used in order to prevent overfitting. It keeps the weights small and avoids exploding gradients.
 - Model architecture: the model is adjusted so that there are not too many parameters to train. As the training size is rather limited, a relatively simple model might have better performance than complicated ones. There are in total 9016 parameters in the final model.
 - Early Stopping is used to stop training if the validation loss does not drop anymore. This is also to prevent overfitting issues.

Step 3. Check the test accuracy versus the number of training images used

Train the model with different training size and plot the corresponding test accuracy:

- Training sizes used are 10, 20, 40, 80, 160, 320, 640, 1280, 1500 for each class to ensure balanced class. Hence, the total training sizes are 40, 80, 160, 320, 640, 1280, 2560, 5120, 6000.
- The validation dataset is unchanged throughout the training process, which is the other 30% of the dataset separated in Step 2.
- The accuracy is calculated as the total correct predictions over the number of test cases used.
- For each training size, 40 epochs are used with early stopping, where if the validation loss is not decreasing for 10 epochs, training will be halted and the weights from the best model will be retrieved.

- The test accuracy and training size can be plotted as below:



Comments:

As shown in the plot, the test accuracy was about 83% with 40 images, which then decreased and increased back to around 85%.

1. Potential reasons to accuracy drop

Theoretically, the test accuracy should increase with the training size. However, for my model, the accuracy first dropped and then increased. There might be a few reasons leading to this:

- The early stopping patience of 10 is too small.
- The total epoch available is only 40, which can be too low for some training sizes. For example, with training sizes of 160, 320 and 6000, 40 epochs were reached before early stopping was activated. This does correspond to the drops on the test accuracy curve. Hence, further increasing the epoch may help resolve this issue.
- Besides, the accuracy is pretty high with only 40 training images, indicating that with proper model setup, it is possible for neural networks to generate relatively good prediction results.

2. Loss difference between training and validation set

Moreover, during the training process, the training loss was found to be much higher than the validation loss. This is partially due to the usage of 50% dropout, where 50% of the nodes were not connected when calculating training accuracy, but fully connected when calculating the validation accuracy. Usage of batch normalization may also contribute to the difference. During the training process, the batch normalization uses mean and variances of the given input batch, which might be changing. However, under evaluation mode, it uses the running mean and variance, both of which reflect properties of the whole training set much better.