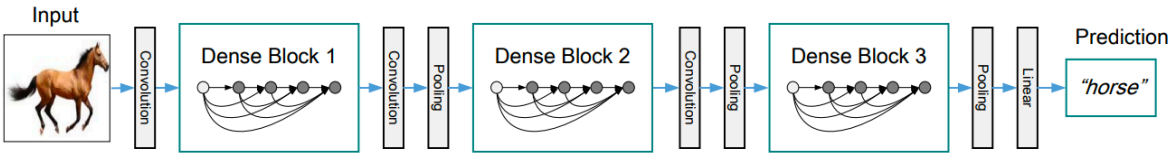
Report for CVDL HW01 - Scene classification

张孝帅 Xiaoshuai Zhang, 1500010716

jet@pku.edu.cn

1. Model Overview

The proposed method is deep convolutional neural network based. More specifically, the DenseNet161 is adopted as our base model. We've also tried ResNet50, ResNet152 and VGG19. Simple comparisons are illustrated later.



The DenseNet is an effective architecture for the task of image classification which appears as an Oral in CVPR 2017. The ideas are quite simple and kind of similar to the ResNet. Shortcut are used to ease gradient vanishing. Moreover, the convolutional layers are densely connected in a Dense Block, which means all outputs of preceding convolutional layers are *concatecated* as input of the current layer.

The observation of DenseNet is that the architecture can efficiently re-use the features extracted from different depths. And by fully exploit the features, the DenseNet reduces the number of parameters while keeping the performance (compared to ResNet).

2. Dataset

The provided training set is divided into a real training set and a validation set. An equalized division is performed. The training set consists of 46820 images, and the validation set consists of 9179 images. The ratio (val / all) is 0.164.

3. Experiments

All experiments are performed on 4 GTX Titan Xps. Optimizer parameters are sophisticatedly adjusted. In most cases, the Adam optimizer is used, and the initial learning rate is 0.001.

Various methods of data augmentation / processing are performed, including:

- Color jittering
    - Brightness ~0.2, Contrast ~0.3, Hue ~0.1, Saturation ~0.1.
    - Random grayscale with possiblity = 0.1
- Random image rotation ~10 deg
- Random horizontal flip with possibility = 0.5
- Random resized crop
    - Random scaling (0.08, 1.0), Random aspect ratio (3/4, 4/3)

    - Finnally resized to (224, 224)
- Normalization using imagenet arguments
    - mean = [0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]

Generally, three training methods are tested:

1. Train from scratch. Weights are initialized using the Xavier initializer
2.
    - Loading feature weights from ImageNet pretrained models.
    - Fix the feature component and tune the classification component (*i.e* the fully connected layer).
    - Tuning the whole network, with smaller learning rate for the classification component.
3. Similar to 2., the weights are loaded from Place365 pretrained models instead.

4. Results

Model	**acc**	**top-3 acc**
DenseNet161, from Place365	**0.8316**	**0.9540**
ResNet50, from Place365	0.7298	0.8952
ResNet152, from ImageNet	0.7649	0.9201
VGG19, from ImageNet	0.6776	-

The experiments are done systematically. But many results are deleted by mistake. So only small part of the results are illustrated here.

5. Discussions

The model have achieved pretty good performance, but there're still limitations:

1. The model is huge and resource-consuming, so that inferences on CPUs or a single GPU is slow. In future works, I plan to use latest techniques of compression and transfer learning to optimize the model.
2. The following picture in the test set is a failure case for our model:

> The predicted top-3 labels are skating_rink, igloo/ice_engraving, art_room, which are all wrong obviously. (I think the problem is in the training set though) Possible solutions include enlarge the training set, and maybe we can generate cartoon images using cycleGAN (high-level data augmentation).

6. Notes for the Codes

Dependencies

python (>= 3.5), with

- PIL (>= 4.1.1),
- tqdm (>= 4.23.0),
- pytorch (>= 0.3.0),
- torchvision (>= 0.2.0).

Testing

The trained weight file is under `./checkpoints`, which reaches 83.2% acc and 95.4% top-3 acc on validation set.

Single Image Testing

To perform classification on a single image:

```
python test.py -c [CHECKPOINT_PATH] -s [IMAGE_PATH]
```

Image Directory Testing

To perform classification on an image directory:

```
python test.py -c [CHECKPOINT_PATH] -i [IMAGE_DIR] -o [OUTPUT_FILE]
```

The output file will be in the format of:

```
# Filename      # Top-3 classes
test/051247.jpg 15 11 4
test/012087.jpg 49 58 50
...
```

Addictional Options

```
--cpu             Test on CPUs rather than on GPUs
--parallel, -p    Run batches parallel (using nn.DataParallel)
--tenctop, -t     Run tencrop test
```