# Assignment 1 Report

## DDA4220 Deep Learning and Applications

**Chuqiao Feng (120090272)**
School of Data Science
The Chinese University of Hong Kong, Shenzhen
`120090272@link.cuhk.edu.cn`

## Contents

## 1 Solution

The basic model in this paper is based on the simple CNN with 3 convolutional layers provided in the question file, whose purpose is to classify the image data in the flower dataset into five categories: 'daisy', 'dandelion', 'rose', 'sunflower', and 'tulip'. For the solution required in the 'graded function' region of the question, we can divide the question into 'neural network modeling', 'dataset preparation', and 'model training' to solve the problem.

The structure of the first two convolutional layers $Convolution - ReLU - MaxPool$ has been given in the topic. To improve the accuracy of the classification model, I added Batchnormal layers in each CNN layer to rescale the previous data input to achieve better results. In addition, in this solution, I use AdaptiveAvgpool as the pooling layer at the end of the layer before the classifier. For the classifier of this classification model, this solution modifies the MLP given in the question, so that the node does not drop directly from 4096 to 5, but adds an intermediate Linear layer to make it have a decreasing process. The following is the specific process of this neural network implementation.

## 2 Experiments

### 2.1 Data

The sample dataset can be divided into 5 classes, the details of the data are in the table below.

| Class | Amount |
|---|---|
| Daisy | 588 |
| Dandelion | 556 |
| Rose | 583 |
| Sunflower | 536 |
| Tulip | 585 |

The second question of the solution uses the same dataset processing as in the first question, dividing the image data into a training set and a validation set according to 8:2 and organizing them into ImageNet format for neural network training. In order to convert the images from $.jpg$ format to a tensor format that can be processed by Pytorch and the neural network, the torch.utils.data.DataLoader and torchvision.datasets function from the Pytorch library are used in this solution to process the datasets separately, and in each epoch of training, the images are shuffled to achieve better training results.

## 2.2 Evaluation method

This solution uses a precision measure to evaluate the accuracy of the classification model. We use Top-1 precision in this solution which is defined as the percentage of test samples where the predicted label of the model is the same as the ground truth label. Which means:

$$acc = \frac{numberofprediction = truth}{numberoftruth}$$

## 2.3 Experimental details

### model structure

Each Convolution Neural Network consists a structure of $Conv - ReLU - BatchNorm2d - MaxPool$ which is as in the picture below:

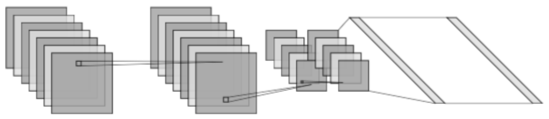


Figure 1: Part of Conv

and based on the requirements in the sample code, I also added BatchNormal layers after each layer of the ReLu function.

For example, when input $x$ is in size $N * D$, then in each channel:

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}, Per-channelmean, ShapeD$$
$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij}-\mu_j)^2, Per-channelstd, ShapeD$$
$$\hat{x_{ij}} = \frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\epsilon}}, Normalizedx, ShapeN * D$$

During testing, batchnorm becoms a linear operator, which can be fused with convolution layer, which can makes deep network much easier to train, allows higher learning rates and faster convergence.

Also I added the 4th Conv layer after that, because I found it has better performance in experiment. So the logic structure of this model is like:

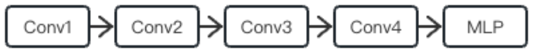


Figure 2: Structure

In addition, after Conv4 before MLP, AvgPool is added as the last pool layer, whose Adaptive function can help to calculate the size of kernel and the value of step size. Adaptive can automatically help us calculate the size of the kernel and the step size of each move when the size of the input and output data is given. Moreover, in the MLP classifier, I added dropout layer with probability 0.4 between layers, which can reduce the risk of model overfitting.

**Learning strategy and other parameters**

In this solution we use the Stochastic gradient decent optimizer with initial learning rate of 0.01, momentum 0.9 and 0.0001 weight decay. Also, instead of stepLR scheduling strategy, I choose Cosine Annealing scheduler for it has better performance. And we use the cross-entropy as loss function. After several times of experiment, I choose batch size=128 and epoch number = 50 in the final training.

## 2.4 Results and Analysis

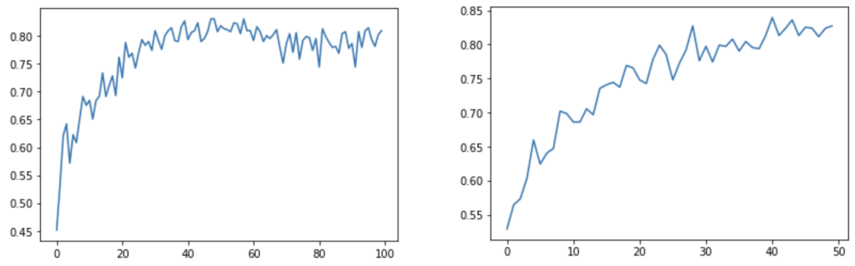
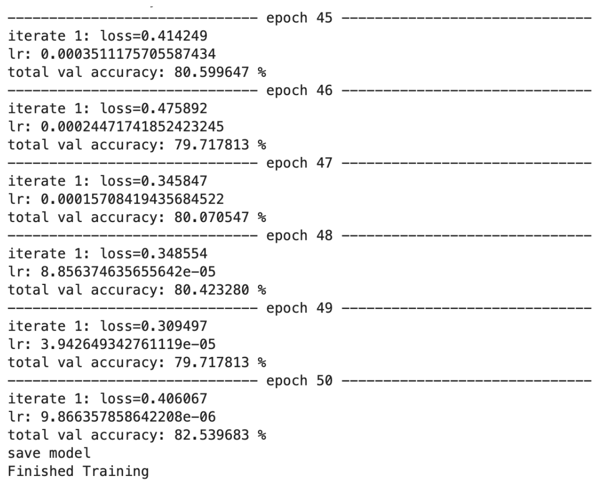


Figure 3: figure title

After many experiments, I found that the best performance was achieved when the learning rate was 0.01. However, when 100 epochs were taken for the experiments, it was found that the model tended to overfit from epoch=50 onwards, and the final training results of the model were obtained as follows.



```
------------------------------ epoch 45 ------------------------------
iterate 1: loss=0.414249
lr: 0.0003511175705587434
total val accuracy: 80.599647 %
------------------------------ epoch 46 ------------------------------
iterate 1: loss=0.475892
lr: 0.00024471741852423245
total val accuracy: 79.717813 %
------------------------------ epoch 47 ------------------------------
iterate 1: loss=0.345847
lr: 0.00015708419435684522
total val accuracy: 80.070547 %
------------------------------ epoch 48 ------------------------------
iterate 1: loss=0.348554
lr: 8.856374635655642e-05
total val accuracy: 80.423280 %
------------------------------ epoch 49 ------------------------------
iterate 1: loss=0.309497
lr: 3.942649342761119e-05
total val accuracy: 79.717813 %
------------------------------ epoch 50 ------------------------------
iterate 1: loss=0.406067
lr: 9.866357858642208e-06
total val accuracy: 82.539683 %
save model
Finished Training
```

Figure 4: Output sample