

# MSBD 6000B Project II Report

---

**Student Name:** LIU Yan Yun      **Student Id:** 20384933

## I. Introduction

---

In this project, I tried two different methods to classify flower photos. Firstly, I trained and evaluated a CNN model with training and validation sets, the validation accuracy is about 78%, which is not good enough. Training an entire CNN model from scratch could not achieve a satisfactory performance since we only had limited data (2500+ for training and 500+ for validation). So I considered to use a pretrained model with a large scale of data, and treated the last layer output as initialization for my prediction model. With the help of this transfer learning workflow, my validation accuracy increased to 90%.

## II. Stage 1: Train a random initialization CNN

---

### a. Pre-processing

**Resize image:** The row input image is too large and not unique. I need to resize those images with a fixed size, and the size of image should not be too large so that my laptop can handle.

**Normalization:** Before training, normalize all image arrays into range 0-1

**One hot encoding:** For labels, use one hot encoding to transform them into vectors.

### b. Train CNN model

The model structure is shown below:

1st Part: Convolutional layer (32 features and 3\*3 filter size, SAME padding) with batch normalization and ReLU activation. No pooling layer, no drop-out layer.

2nd Part: Convolutional layer (32 features and 3\*3 filter size, SAME padding) with batch normalization and ReLU activation. Has one max pooling layer with pool\_size = 2, and drop-out rate is 0.25.

3rd Part: Convolutional layer (64 features and 3\*3 filter size, SAME padding) with batch normalization and ReLU activation. No pooling layer, and drop-out rate is 0.25.

4th Part: Convolutional layer (64 features and 3\*3 filter size, SAME padding) with batch normalization and ReLU activation. Has one max pooling layer with pool\_size = 2, and drop-out rate is 0.25.

5th Part: Flatten fully connected layer with 512 units, batch normalization layer and ReLU activation. Drop-out rate is 0.5

6th: Soft-max layer.

### c. Some tricks I use

**Batch normalization:** Adding batch normalization layer improved my validation accuracy.

**Alternate pooling layer:** I only used max pooling in layers with first two even part.

**Drop-out :** I set a lower drop-out rate to avoid overfitting at the first 4 parts, but in the dense fully connected layer, the drop-out rate should be a little higher since the features in this stage is more meaningful and worthy.

#### d. Validation accuracy

the final validation accuracy is 78%

### III. Stage 2: Transfer learning

---

Since the performance for CNN model is not good enough, then I considered to use transfer learning method (see reference) to make some improvement. I use the code provided by tensorflow's Github, loading a pre-trained Inception v3 model.

#### a. Pre-processing

No special preprocessing stage in my transfer learning model, just follow Inception v3 model workflow.

#### b. Train the model

1. Load pre-trained Inception v3 model and flower dataset
2. Calculates the bottleneck values for each image
3. Train a new model on the top of bottleneck values
4. Model evaluation and prediction

#### d. Validation accuracy

the final validation accuracy is 90%

### IV. Model selection and evaluation

---

You can use my transfer learning model prediction output for grading since it achieved higher validation accuracy. Meanwhile, I list the evaluation result on validation set for those two methods:

Keras CNN:

|             | precision | recall | f1-score | support |
|-------------|-----------|--------|----------|---------|
| class 0     | 0.87      | 0.82   | 0.84     | 110     |
| class 1     | 0.76      | 0.80   | 0.78     | 122     |
| class 2     | 0.65      | 0.73   | 0.69     | 93      |
| class 3     | 0.80      | 0.91   | 0.85     | 103     |
| class 4     | 0.82      | 0.64   | 0.72     | 122     |
| avg / total | 0.78      | 0.78   | 0.78     | 550     |

Transfer learning with Tensorflow:

|             | precision | recall | f1-score | support |
|-------------|-----------|--------|----------|---------|
| class 0     | 0.86      | 0.91   | 0.88     | 122     |
| class 1     | 0.85      | 0.83   | 0.84     | 93      |
| class 2     | 0.91      | 0.93   | 0.92     | 122     |
| class 3     | 0.95      | 0.86   | 0.90     | 103     |
| class 4     | 0.94      | 0.95   | 0.95     | 110     |
| avg / total | 0.90      | 0.90   | 0.90     | 550     |

## V. Reference

---

1. Keras: [https://github.com/fchollet/keras/blob/master/examples/cifar10\\_cnn.py](https://github.com/fchollet/keras/blob/master/examples/cifar10_cnn.py)
2. Transfer learning: [https://www.tensorflow.org/tutorials/image\\_retraining](https://www.tensorflow.org/tutorials/image_retraining)