꽃 이미지 분류 신경망

꽃 데이터는 320 * 240 pixel 을 가진 4323 장의 꽃(daisy :769, dandelion : 1055, rose : 784, sumflower : 734, tulip : 984)으로 구성되어 있다. 다층 퍼셉트론을 통해 꽃 이미지를 분석하여 유의미한 결과 값을 내는 것이 이번 실습의 목표이다.

코드

```
%run ../chap05/mathutil.ipynb

np.random.seed(1234)
# 난수 발생 패턴 초기화. 이를 통해 동일한 Train, Test set 에 의한 결과가 출력되어 평가 기준을 적용시키는데 일관성을 얻을 수 있다.
def randomize(): np.random.seed(time.time())
```

```
class Model(object):
    def __init__(self, name, dataset):
        self.name = name
        self.dataset = dataset
        self.is_training = False # 학습 중 여부 판단
        # hasattr : 가진 속성이면 True, 아니면 False 를 반환하는 함수
        if not hasattr(self, 'rand_std'):
            self.rand_std = 0.030 # parameter 초기화에 이용

    def __str__(self):
        return '{}/{}'.format(self.name, self.dataset)

    # 전체 과정을 실행시키는 main 함수와 같은 역할
    def exec_all(self, epoch_count=10, batch_size=10, learning_rate=0.001,
                 report=0, show_cnt=3):
        self.train(epoch_count, batch_size, learning_rate, report) # 학습
        self.test() # 평가
        if show_cnt > 0: self.visualize(show_cnt) # 시각화 출력 대상이 0 이상일때만 시각화
```

```
class MlpModel(Model):
    def __init__(self, name, dataset, hconfigs):
        super(MlpModel, self).__init__(name, dataset)
        self.init_parameters(hconfigs)
        '''
        super 명령을 이용해 기반 클래스인 Model 클래스를 찾고
        (1. super 명령을 이용하여 기본 클래스 명시할 필요 없게, 2. (다중) 상속)
        그 Model 클래스의 객체 초기화 함수를 호출하여 name 과 dataset 값을 저장한다.
        또한 init_parameters() 메소드를 호출하여 신경망이 이용할 파라미터 준비
        클래스 선언 외부에서 함수를 정의하고 이를 method 로 등록하는 방식을 이용하여
```

```
        프로그램의 가독성 향상
        '''
```

```
def mlp_init_parameters(self, hconfigs):
    self.hconfigs = hconfigs
    self.pm_hiddens = []

    #dataset 객체의 속성값으로부터 입출력 벡터 크기 등의 정보를 얻는다.
    prev_shape = self.dataset.input_shape

    for hconfig in hconfigs:
        pm_hidden, prev_shape = self.alloc_layer_param(prev_shape, hconfig)
        self.pm_hiddens.append(pm_hidden) # 생성된 파라미터를 객체 변수로 저장

    output_cnt = int(np.prod(self.dataset.output_shape)) # 배열 요소의 곱 반환
    self.pm_output, _ = self.alloc_layer_param(prev_shape, output_cnt) # 가중치와 편향
파라미터 쌍 생성

def mlp_alloc_layer_param(self, input_shape, hconfig):
    input_cnt = np.prod(input_shape)
    output_cnt = hconfig

    weight, bias = self.alloc_param_pair([input_cnt, output_cnt])

    return {'w':weight, 'b':bias}, output_cnt

def mlp_alloc_param_pair(self, shape):
    weight = np.random.normal(0, self.rand_std, shape)
    bias = np.zeros([shape[-1]])
    return weight, bias

# 코드에서 정의된 함수들을 클래스의 멤버함수(메소드)로 등록한다.
# 정의된 함수 이름과 등록되는 메소드 이름이 서로 다른데 이처럼 두 이름을 독립적으로 부여하면
# 파생 클래스에서 메소드를 재정의할 때 함수 이름을 달리 할 수 있어서
# 프로그램의 가독성이 높아진다.
MlpModel.init_parameters = mlp_init_parameters
MlpModel.alloc_layer_param = mlp_alloc_layer_param
MlpModel.alloc_param_pair = mlp_alloc_param_pair
```

```
def mlp_model_train(self, epoch_count=10, batch_size=10, ₩
                    learning_rate=0.001, report=0):
    self.learning_rate = learning_rate

    batch_count = int(self.dataset.train_count / batch_size)
    time1 = time2 = int(time.time())
```

```
    if report != 0:
        print('Model {} train started:'.format(self.name))

    for epoch in range(epoch_count):
        costs = []
        accs = []
        self.dataset.shuffle_train_data(batch_size*batch_count) # 데이터 뒤섞기. 학습 데이터
접근 방법을 변형시키기 편하다
        for n in range(batch_count):
            trX, trY = self.dataset.get_train_data(batch_size, n)
            cost, acc = self.train_step(trX, trY)
            costs.append(cost)
            accs.append(acc)

        if report > 0 and (epoch+1) % report == 0:
            vaX, vaY = self.dataset.get_validate_data(100) # 검증 데이터
            acc = self.eval_accuracy(vaX, vaY)
            time3 = int(time.time())
            tm1, tm2 = time3-time2, time3-time1
            self.dataset.train_prt_result(epoch+1, costs, accs, acc, tm1, tm2)
            time2 = time3

    tm_total = int(time.time()) - time1
    print('Model {} train ended in {} secs:'.format(self.name, tm_total))

MlpModel.train = mlp_model_train
```

```
def mlp_model_test(self): # 평가 메소드
    teX, teY = self.dataset.get_test_data()
    time1 = int(time.time())
    acc = self.eval_accuracy(teX, teY)
    time2 = int(time.time())
    self.dataset.test_prt_result(self.name, acc, time2-time1)

MlpModel.test = mlp_model_test
```

```
def mlp_model_visualize(self, num): # 시각화 메소드
    print('Model {} Visualization'.format(self.name))
    deX, deY = self.dataset.get_visualize_data(num)
    est = self.get_estimate(deX)
    self.dataset.visualize(deX, est, deY)

MlpModel.visualize = mlp_model_visualize
```

```
def mlp_train_step(self, x, y):
    self.is_training = True

    output, aux_nn = self.forward_neuralnet(x) # 순전파 처리
    loss, aux_pp = self.forward_postproc(output, y)
    accuracy = self.eval_accuracy(x, y, output) # 보고에 사용될 정확도

    G_loss = 1.0
    G_output = self.backprop_postproc(G_loss, aux_pp) # 역전파 처리
    self.backprop_neuralnet(G_output, aux_nn)

    self.is_training = False # 검증 과정은 비학습 상태에서 이루어져야 하므로 False

    return loss, accuracy

MlpModel.train_step = mlp_train_step
```

```
def mlp_forward_neuralnet(self, x):
    hidden = x
    aux_layers = []

    for n, hconfig in enumerate(self.hconfigs):
        hidden, aux = self.forward_layer(hidden, hconfig, self.pm_hiddens[n]) # Hidden Layer
        aux_layers.append(aux)

    output, aux_out = self.forward_layer(hidden, None, self.pm_output) # 출력 Layer

    return output, [aux_out, aux_layers]

def mlp_backprop_neuralnet(self, G_output, aux):
    aux_out, aux_layers = aux # 역전파용 보조 정보

    G_hidden = self.backprop_layer(G_output, None, self.pm_output, aux_out)

    for n in reversed(range(len(self.hconfigs))):
        hconfig, pm, aux = self.hconfigs[n], self.pm_hiddens[n], aux_layers[n]
        G_hidden = self.backprop_layer(G_hidden, hconfig, pm, aux)

    return G_hidden

MlpModel.forward_neuralnet = mlp_forward_neuralnet
MlpModel.backprop_neuralnet = mlp_backprop_neuralnet
```

```
def mlp_forward_layer(self, x, hconfig, pm):
    y = np.matmul(x, pm['w']) + pm['b']
```

```
    if hconfig is not None: y = relu(y) # 비선형 활성화 함수는 은닉 계층에만 적용되도록
    return y, [x,y]

def mlp_backprop_layer(self, G_y, hconfig, pm, aux):
    x, y = aux
     # 비선형 활성화 함수는 은닉 계층에만 적용되도록
    if hconfig is not None: G_y = relu_derv(y) * G_y

    g_y_weight = x.transpose()
    g_y_input = pm['w'].transpose()

    G_weight = np.matmul(g_y_weight, G_y)
    G_bias = np.sum(G_y, axis=0)
    G_input = np.matmul(G_y, g_y_input)

    pm['w'] -= self.learning_rate * G_weight
    pm['b'] -= self.learning_rate * G_bias

    return G_input

MlpModel.forward_layer = mlp_forward_layer
MlpModel.backprop_layer = mlp_backprop_layer
```

```
# 손실 함수를 계산하는 과정을 메소드로 처리하게 하여 확장성 증가.
def mlp_forward_postproc(self, output, y):
    loss, aux_loss = self.dataset.forward_postproc(output, y)
    extra, aux_extra = self.forward_extra_cost(y)
    return loss + extra, [aux_loss, aux_extra]

def mlp_forward_extra_cost(self, y):
    return 0, None
MlpModel.forward_postproc = mlp_forward_postproc
MlpModel.forward_extra_cost = mlp_forward_extra_cost
```

```
def mlp_backprop_postproc(self, G_loss, aux):
    aux_loss, aux_extra = aux
    self.backprop_extra_cost(G_loss, aux_extra)
    G_output = self.dataset.backprop_postproc(G_loss, aux_loss)
    return G_output

def mlp_backprop_extra_cost(self, G_loss, aux):
    pass

MlpModel.backprop_postproc = mlp_backprop_postproc
MlpModel.backprop_extra_cost = mlp_backprop_extra_cost
```

```
def mlp_eval_accuracy(self, x, y, output=None):
    if output is None:
        output, _ = self.forward_neuralnet(x)
    accuracy = self.dataset.eval_accuracy(x, y, output)
    return accuracy

MlpModel.eval_accuracy = mlp_eval_accuracy
```

```
def mlp_get_estimate(self, x):
    output, _ = self.forward_neuralnet(x)
    estimate = self.dataset.get_estimate(output)
    return estimate

MlpModel.get_estimate = mlp_get_estimate
```

보조 기능이 들어있는 mathutil.ipynb 파일을 통해 dataset.ipynb 의 클래스와 파생 클래스들이 이용할 파이썬 모듈을 사용할 수 있게 하고 추가로 필요한 함수를 정의한다.

dataset.ipynb 에는

- 미니배치 학습 데이터를 공급하는 메소드
- 학습용 데이터를 뒤섞어주는 메소드
- 평가 데이터 공급 메소드
- 검증 데이터 일부를 반환하는 메소드
- 시각화 지원 메소드
- 데이터를 뒤섞는 메소드
- 후처리 순전파 처리 지원 메소드(손실 함숫값 계산 등)
- 후처리 역전파 처리 지원 메소드(손실 기울기 계산 등)
- 정확도 계산 메소드
- 추정 결과 시각화 변환 메소드
- 로그 출력 메소드(중간 결과 출력)

등이 포함되어 있다.

이를 통해 이전에 배웠던 전복 고리 수 추정 신경망, 텐서의 펄서 여부 판정 신경망, 철판의 불량 판정 여부 신경망 등에도 확장시킬 수 있었다.

꽃 이미지 분류에는 dataset.flowers.ipynb 를 이용하여 앞서 사용된 dataset 을 선언하여 사용한다. 추가로 flower dataset 의 클래스, 객체 초기화, 시각화 메소드를 정의하고 mlp_model_test.ipynb 를 이용하여 학습 결과를 확인할 수 있었다.

[Test1]

```
fd = FlowersDataset()
fm = MlpModel('flowers_model_1', fd, [10])
fm.exec_all(epoch_count=10, report=2)
```

```
Model flowers_model_1 train started:
    Epoch 2: cost=1.774, accuracy=0.243/0.260 (4/4 secs)
    Epoch 4: cost=1.754, accuracy=0.243/0.270 (4/8 secs)
    Epoch 6: cost=1.738, accuracy=0.243/0.270 (5/13 secs)
    Epoch 8: cost=1.724, accuracy=0.243/0.270 (4/17 secs)
    Epoch 10: cost=1.713, accuracy=0.243/0.280 (5/22 secs)
Model flowers_model_1 train ended in 22 secs:
Model flowers_model_1 test report: accuracy = 0.239, (0 secs)

Model flowers_model_1 Visualization
```

```
추정확률분포 [17,20,10,17,17,19] => 추정 dandelion : 정답 tulip => X
추정확률분포 [17,20,10,17,17,19] => 추정 dandelion : 정답 sunflower => X
추정확률분포 [17,20,10,17,17,19] => 추정 dandelion : 정답 rose => X
```

Hidden Layer : 1
Node : 10
Epoch : 10
Accuracy : 0.239 (사실상 의미 없는 결과, 추정 확률 분포도 마찬가지로 의미가 없다.)
예시 : 0/3 정답

[Test2] – Test 1 에서 Hidden Layer 수를 증가시키고 Node 수를 조정하였다.

```
fm2 = MlpModel('flowers_model_2', fd, [30, 10])
fm2.exec_all(epoch_count=10, report=2)
```

```
Model flowers_model_2 train started:
    Epoch 2: cost=1.604, accuracy=0.248/0.270 (13/13 secs)
    Epoch 4: cost=1.414, accuracy=0.370/0.360 (14/27 secs)
    Epoch 6: cost=1.336, accuracy=0.410/0.400 (13/40 secs)
    Epoch 8: cost=1.291, accuracy=0.438/0.400 (14/54 secs)
    Epoch 10: cost=1.262, accuracy=0.445/0.380 (13/67 secs)
Model flowers_model_2 train ended in 67 secs:
Model flowers_model_2 test report: accuracy = 0.370, (1 secs)

Model flowers_model_2 Visualization
```

```
추정확률분포 [57,40, 0, 2, 0, 0] => 추정 daisy : 정답 dandelion => X
추정확률분포 [35,47, 0, 5, 4, 9] => 추정 dandelion : 정답 dandelion => O
추정확률분포 [ 1, 1, 0,74, 0,24] => 추정 rose : 정답 rose => O
```

Hidden Layer : 2
Node : Hidden Layer 각각[30, 10]
Epoch : 10
Accuracy : 0.370 (사실상 의미 없는 결과, 추정 확률 분포도 마찬가지로 의미가 없다.)
예시 : 2/3 정답

[Test3] – Test 2 에서 Hidden Layer 의 Node 수를 조정하고, Epoch 를 50 으로 크게 늘였다.

```
fm3= MlpModel('flowers_model_3', fd, [40, 20])
fm3.exec_all(epoch_count=50, report=5)
```

```
Model flowers_model_3 train started:
    Epoch 5: cost=1.458, accuracy=0.352/0.440 (39/39 secs)
    Epoch 10: cost=1.274, accuracy=0.434/0.420 (41/80 secs)
    Epoch 15: cost=1.177, accuracy=0.497/0.410 (39/119 secs)
    Epoch 20: cost=1.078, accuracy=0.541/0.450 (40/159 secs)
    Epoch 25: cost=0.989, accuracy=0.583/0.430 (40/199 secs)
    Epoch 30: cost=0.924, accuracy=0.614/0.440 (39/238 secs)
    Epoch 35: cost=0.849, accuracy=0.645/0.500 (39/277 secs)
    Epoch 40: cost=0.777, accuracy=0.689/0.440 (39/316 secs)
    Epoch 45: cost=0.719, accuracy=0.713/0.530 (40/356 secs)
    Epoch 50: cost=0.706, accuracy=0.721/0.470 (40/396 secs)
Model flowers_model_3 train ended in 396 secs:
Model flowers_model_3 test report: accuracy = 0.440, (0 secs)

Model flowers_model_3 Visualization
```

```
추정확률분포 [46,23, 0, 2,17,12] => 추정 daisy : 정답 dandelion => X
추정확률분포 [ 0, 3, 0, 1,75,21] => 추정 sunflower : 정답 sunflower => O
추정확률분포 [59,18, 0, 4, 6,13] => 추정 daisy : 정답 rose => X
```

Hidden Layer : 2
Node : Hidden Layer 각각 [40, 20]
Epoch : 50
Accuracy : 0.370
예시 : 2/3 정답

[Test4] – Test 3 에서 Epoch 수를 늘이고 모델을 바꿔서 학습해 보았다.

```
fm4= MlpModel('flowers_model_2', fd, [40, 20])
fm4
.exec_all(epoch_count=100, report=5)
```

```
Model flowers_model_2 train started:
    Epoch 5: cost=1.332, accuracy=0.412/0.490 (40/40 secs)
    Epoch 10: cost=1.237, accuracy=0.459/0.300 (40/80 secs)
    Epoch 15: cost=1.163, accuracy=0.509/0.420 (40/120 secs)
    Epoch 20: cost=1.073, accuracy=0.546/0.460 (40/160 secs)
    Epoch 25: cost=1.008, accuracy=0.585/0.440 (39/199 secs)
    Epoch 30: cost=0.921, accuracy=0.632/0.500 (39/238 secs)
    Epoch 35: cost=0.861, accuracy=0.659/0.440 (40/278 secs)
    Epoch 40: cost=0.835, accuracy=0.666/0.480 (39/317 secs)
    Epoch 45: cost=0.749, accuracy=0.703/0.320 (40/357 secs)
    Epoch 50: cost=0.702, accuracy=0.726/0.380 (38/395 secs)
    Epoch 55: cost=0.656, accuracy=0.749/0.410 (42/437 secs)
    Epoch 60: cost=0.637, accuracy=0.763/0.460 (41/478 secs)
    Epoch 65: cost=0.580, accuracy=0.778/0.470 (39/517 secs)
    Epoch 70: cost=0.605, accuracy=0.772/0.360 (41/558 secs)
    Epoch 75: cost=0.496, accuracy=0.815/0.390 (42/600 secs)
    Epoch 80: cost=0.529, accuracy=0.805/0.510 (40/640 secs)
    Epoch 85: cost=0.480, accuracy=0.822/0.480 (40/680 secs)
    Epoch 90: cost=0.463, accuracy=0.825/0.390 (40/720 secs)
    Epoch 95: cost=0.439, accuracy=0.845/0.530 (39/759 secs)
    Epoch 100: cost=0.501, accuracy=0.823/0.450 (39/798 secs)
Model flowers_model_2 train ended in 798 secs:
Model flowers_model_2 test report: accuracy = 0.447, (0 secs)

Model flowers_model_2 Visualization
```

```
추정확률분포 [ 2,97, 0, 0, 0, 0] => 추정 dandelion : 정답 dandelion => O
추정확률분포 [13,15, 0,14,37,21] => 추정 sunflower : 정답 dandelion => X
추정확률분포 [ 1, 1, 0, 1,81,15] => 추정 sunflower : 정답 sunflower => O
```

Hidden Layer : 2
Node : Hidden Layer 각각 [40, 20]
Epoch : 100
Accuracy : 0.447
예시 : 2/3 정답

[Test5] – Test 4 에서 Epoch 수 감소, Hidden Layer 를 1 층 늘여서 학습해보았다.

```
fm5= MlpModel('flowers_model_2', fd, [20, 10, 5])
fm5.exec_all(epoch_count=30, report=3)
```

```
Model flowers_model_2 train started:
    Epoch 3: cost=1.495, accuracy=0.311/0.260 (17/17 secs)
    Epoch 6: cost=1.429, accuracy=0.343/0.390 (17/34 secs)
    Epoch 9: cost=1.275, accuracy=0.434/0.450 (18/52 secs)
    Epoch 12: cost=1.216, accuracy=0.463/0.460 (16/68 secs)
    Epoch 15: cost=1.171, accuracy=0.488/0.520 (16/84 secs)
    Epoch 18: cost=1.134, accuracy=0.519/0.490 (16/100 secs)
    Epoch 21: cost=1.080, accuracy=0.546/0.460 (17/117 secs)
    Epoch 24: cost=1.039, accuracy=0.559/0.430 (16/133 secs)
    Epoch 27: cost=0.990, accuracy=0.588/0.490 (16/149 secs)
    Epoch 30: cost=0.962, accuracy=0.603/0.350 (16/165 secs)
Model flowers_model_2 train ended in 165 secs:
Model flowers_model_2 test report: accuracy = 0.419, (1 secs)

Model flowers_model_2 Visualization
```



```
추정확률분포 [21,18, 1,17,24,18] => 추정 sunflower : 정답 daisy => X
추정확률분포 [ 2, 8, 0, 1,84, 5] => 추정 sunflower : 정답 sunflower => O
추정확률분포 [ 1, 0, 0,63, 0,36] => 추정 rose : 정답 tulip => X
```

Hidden Layer : 3
Node : Hidden Layer 각각 [20, 10, 5]
Epoch : 30
Accuracy : 0.419
예시 : 1/3 정답

결론

Epoch 수를 늘이고, Hidden Layer 의 수 증가, 퍼셉트론 개수 조정 등의 하이퍼파라미터 튜닝을 이용해서 0.5 이상의 정확성을 얻기 어려웠다.

파라미터에 비해 데이터의 수가 적어서 제대로 학습이 이루어 지지 않았다. 적은 수의 데이터이고, Hidden Layer 의 수를 적게 구성하였으나, 이미지 데이터가 가지는 정보가 많다 보니 시간이 오래 걸렸다. 양질의 이미지에 데이터의 양이 많다면 학습에 소요되는 시간과 비용이 엄청날 것이라는 짐작이 가능하였다.

다층 퍼셉트론이 이후 배우게 될 CNN 에 비해 많이 뒤쳐진다는 점을 확인할 수 있었다. 이번 실습은 다층 퍼셉트론과 대비되는 CNN(Convolutional Neural Network)의 성능을 측정하는데 Baseline 으로 활용할 수 있다.

난수 발생 패턴을 고정시켜야 Train, Test set 에 의한 결과가 동일하게 나와서 학습이 잘 되었는지를 평가하는데 도움이 된다. 고정시키지 않으면 매번 다른 Train, Test set 에 의한 결과가 출력이 되어 평가 기준을 적용하는데 일관성이 떨어진다.

- 신경망의 파라미터가 많아지면 생기는 문제?
  : 이미지는 데이터 크기가 매우 크다. 이 예제의 데이터만 보아도 pixel size 로 가로의 크기를 320 으로 하고, 세로의 크기를 240 으로 해서 가로 * 세로 * 3(RGB)로만 계산하여도 230,400 이라는 값을 얻을 수 있고 이에 따라 많은 파라미터를 필요로 한다. 그런데, 이렇게 **많은 파라미터를 이용해서 유의미한 결과를 도출해내기 위해서는 많은 양질의 데이터가 필요**한데, 이 예제에서는 고작 4323 장의 사진을 가지고 있다. 파라미터 수보다 적은 데이터 개수로는 좋은 결과를 도출해내는 것이 불가능하다.
- 다층 퍼셉트론 신경망이 이미지 처리에 부적합한 이유?
  : 위에서 볼 수 있지만 크지 않은 이미지 사이즈임에도 저렇게 많은 파라미터를 필요로 하는데, 이미지의 사이즈와 품질이 증가하게 되면 파라미터는 기하급수적으로 증가하게 되고, 그에 따라 데이터도 마찬가지로 많이 필요하게 된다. 하지만 이 4323 개의 사진만 해도 약 230MB 를 차지하는데, 많은 데이터가 있다고 하더라도 학습하는데 시간과 비용이 너무 많이 필요하며, 실제로 그 정도의 많은 양질의 데이터를 얻는 것은 불가능에 가깝다.
- 데이터를 Train/Validation/Test Set 으로 구분짓는 이유?
  : Train Set 과 Test Set 을 나누는 이유는 학습된 데이터를 통해 평가하는 것이 의미가 없을 뿐 아니라 overfitting 을 유발하기 때문이다. 그래서 반드시 Train 과 Test Set 을 나누어 모델을 학습 및 평가해야 한다.
  데이터의 크기가 적어서 Validation Set 을 나누기 힘든 경우에는 Validation Set 을 따로 나누지 않는 경우도 있으나 그럼에도 불구하고 Validation Set 을 사용하는 이유는 성능을 평가할 수 있으며, overfitting 을 방지하는 등 모델을 튜닝하여 품질을 높일 수 있기 때문이다.
  똑같이 성능을 평가하는 Set 이지만 Test Set 과는 차이가 존재하는데, Validation Set 은 Training 의 과정에 관여하여 여러 모델 중 최종 모델을 선정하기 위한 성능 평가에 도움을 주는 반면, Test Set 은 최종 성능을 평가하기 위해 사용되어 Training 과정에 연관되지 않는다.