인공지능 과제 리포트

과제 제목:

5가지 품종별 쌀 이미지 파일의 다중 분류

학번: B711194

이름: 최승혁

**목차**

**1. 과제 개요 3**

**2. 구현 환경 3**

**3. 알고리즘에 대한 설명 4**

**4. 데이터에 대한 설명 6**

4-1. Input feature 6

4-2. Target output 6

**5. 소스코드에 대한 설명 7**

**6. 학습 과정에 대한 설명 8**

6-1. 레이어 하이퍼 파라미터 변동에 따른 실험과 결론 8

6-2. 최종적 CNN 아키텍쳐 결정 10

6-3. validation 데이터 셋을 이용한 최적 하이퍼 파라미터 결정 11

**7. 결과 및 분석 13**

**8. 실행 메뉴얼 15**

**표 목차**

표 1. 구현한 CNN의 hidden layer 정보 5

**그림 목차**

그림 1. 구현한 CNN 4, 10

그림 2. Input data 쌀의 품종 별 예 6

그림 3. 중간발표 모델 1 8

그림 4. 중간발표 모델 2 8

그림 5. 중간발표 모델 1과 모델 2의 정확도 비교. 9

그림 6. 중간발표 모델의 기존 정확도 출력(좌)와 레이어 하이퍼 파라미터를 2배로 한 이후의 정확도 출력(우) 9

그림 7. 사이 무작위 50개 Learning Rate 학습 결과 정확도 그래프 12

그림 8. 마지막 모델의 학습 결과 13

1. 과제 개요

5가지 품종별 쌀 이미지에 대한 다중 분류 모델을 학습한다.

학습한 모델을 사용해서 쌀 이미지를 predict할 때 알맞은 품종으로 분류하는 정확도를 높이는 것을 목표로 한다.

2. 구현 환경

학습의 환경과 사용한 IDE는 Google Colab이다.

구글 코랩의 Python 버전은 3.7.15, GCC 버전은 7.5.0이다.

사용한 Library 및 Package는 다음과 같다.

|  |
| --- |
| * sys, os : 경로 불러오기에 사용 * numpy : 행렬벡터 계산, 머신러닝 모델 구현 * matplotlib : 결과 그래프 플로팅 * pickle : 읽어온 데이터, 여러 numpy array의 보존 * bz2 : pickle의 용량 압축 * collections.OrderedDict : 머신러닝 모델 레이어의 순서 부여 용도 * glob, keras.preprocessing.image.load\_img : 이미지 데이터 읽기 용도 * sklearn.model\_selection : 데이터를 training, validation, test 데이터로 split |

3. 알고리즘에 대한 설명

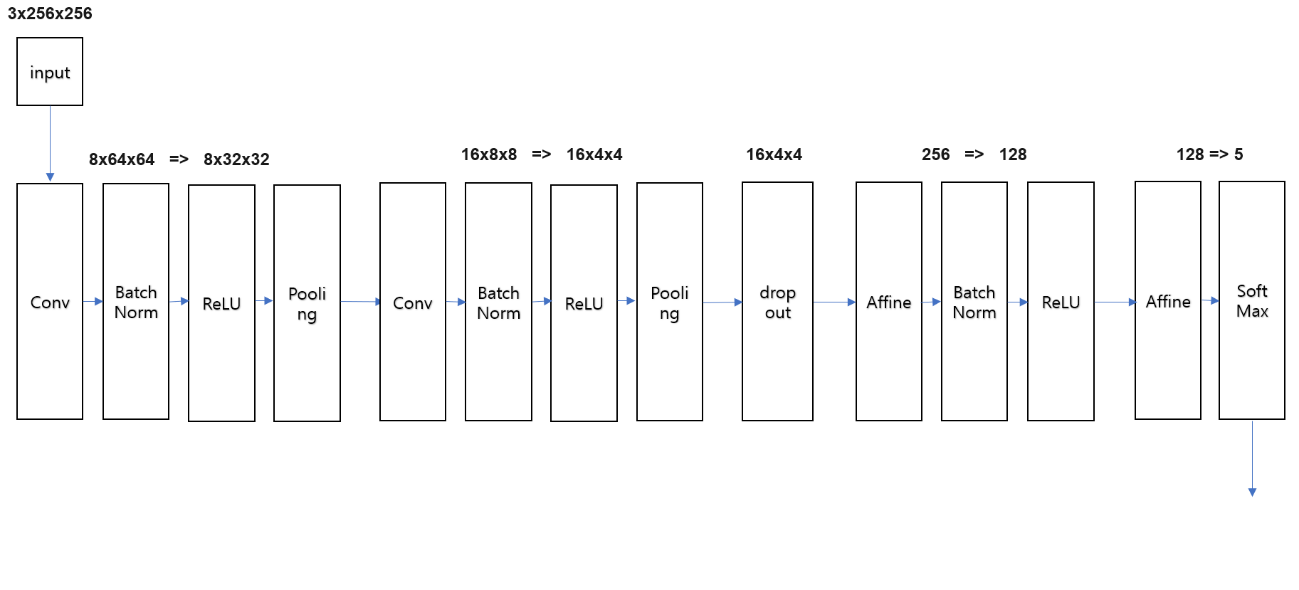
사용한 머신러닝 학습 알고리즘은 역전파를 이용한 Gradient descent이다. 최적화 함수는 Adam을 사용했다.

손실 함수는 교차 엔트로피 오차(Cross Entropy Error)를 사용했다.

데이터 불러오기 및 Convolution layer에서 패딩은 Zero padding을 사용했다.

Pooling layer에서는 Max pooling으로 진행했다.

우리 조의 Machine learning architecture은 총 16층(input+output+14)으로 이루어진 CNN이다.



<그림 1. 구현한 CNN>

그림 1에서 보이듯, 우리 조의 CNN은 총 14층의 hidden layer로 이루어져 있다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **층 이름** | **레이어 하이퍼 파라미터** | **Input size** | **Output size** | **parameter의 수(=node의 수)** |
| Conv1 | filter\_num=8, filter\_size=4\*4,  pad=0, stride=4 | (n, 3, 256, 256) | (n, 8, 64, 64) | 392 (4\*4\*3\*8 +bias) |
| BatchNorm1 | momentum=0.9 | (n, 8, 64, 64) | (n, 8, 64, 64) | 65,536 (8\*64\*64 +8\*64\*64) |
| Relu1 | (ReLU층) | (n, 8, 64, 64) | (n, 8, 64, 64) | 0 |
| Pool1 | pool\_h=2, pool\_w=2, stride=2 | (n, 8, 64, 64) | (n, 8, 32, 32) | 0 |
| Conv2 | filter\_num=16, filter\_size=4\*4,  pad=0, stride=4 | (n, 8, 32, 32) | (n, 16, 8, 8) | 2,064 (4\*4\*8\*16 +bias) |
| BatchNorm2 | momentum=0.9 | (n, 16, 8, 8) | (n, 16, 8, 8) | 2,048 (16\*8\*8 +16\*8\*8) |
| Relu2 | (ReLU층) | (n, 16, 8, 8) | (n, 16, 8, 8) | 0 |
| Pool2 | pool\_h=2, pool\_w=2, stride=2 | (n, 16, 8, 8) | (n, 16, 4, 4) | 0 |
| Dropout | Ratio = 0.3 | (n, 16, 4, 4) | (n, 16, 4, 4) | 0 |
| Affine1 | (완전연결층) | (n, 256) | (n, 128) | 32,896 (256\*128 +bias) |
| BatchNorm3 | momentum=0.9 | (n, 128) | (n, 128) | 256 (128 +128) |
| Relu3 | (ReLU층) | (n, 128) | (n, 128) | 0 |
| Affine2 | (완전연결층) | (n, 128) | (n, 5) | 645 (128\*5 +bias) |
| last\_layer | (Softmax층) | (n, 5) | (n, 5) | 0 |

<표 1. 구현한 CNN의 hidden layer 정보>

표 1에는 각각의 hidden layer의 이름과 레이어 하이퍼 파라미터, 그리고 node에 대한 정보가 나와 있다.

4. 데이터에 대한 설명

4.1 Input Feature

Input data는 250 x250 크기의 JPG 파일 15000개이다. (3000 \* 5개 class)



<그림 2. Input data 쌀의 품종 별 예>

그림 2는 5가지 쌀 종류 이미지를 한 장씩 가져와 예를 보이고 있다. 각 쌀의 품종은 특징적으로 확연히 구분된다.

이 이미지를 Zero padding하여 3\*256\*256 크기로 읽어온다. 따라서 C, H, W = 3, 256, 256이 된다. Input feature의 총 개수는 196,608개이다. (3\*256\*256)

4.2 Target Output

output class는 총 5개이다. 각각 Arborio, Basmati, Ipsala, Jasmine, Karacadag이라는 쌀 품종을 나타낸다.

predict의 softmax를 적용한 결과는 각각 5개 쌀 종류 레이블에 대한 확률값 형태의 원소 5개를 가진 배열로 나타나고, 이를 원-핫-인코딩 한다.

5. 소스코드에 대한 설명

소스 코드의 많은 부분은 ‘밑바닥부터 시작하는 딥러닝’ 교재 깃허브 코드를 가져와 수정했다. (https://github.com/WegraLee/deep-learning-from-scratch/blob/master/)

pickle\_maker.py 파일은 다운로드 받은 이미지 데이터 셋을 피클로 만들어 구글 드라이브 Colab 환경에 간편히 업로드하기 위해 구현했다. Tensorflow 패키지를 불러오지만, 사용하는 함수는 keras.preprocessing.image.load\_img() 뿐이다.

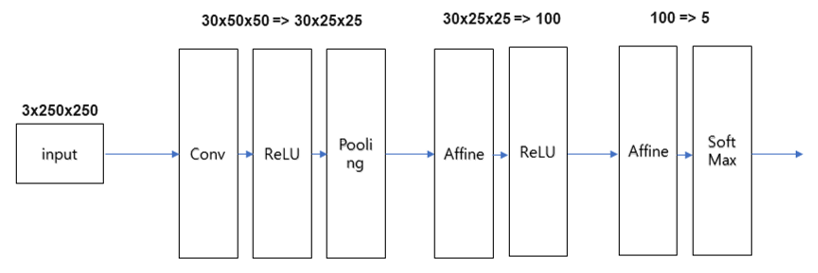
our\_cnn.py 파일의 ConvNet 클래스가 우리 조의 CNN 아키텍쳐이다. weight\_std1~4 변수는 각 ReLU 활성화 함수를 사용할 히든 레이어들의 파라미터 초기 분산값을 He 초기값으로 계산해 미리 결과를 저장한 변수들이다.

trainer.py 파일의 Trainer 클래스는 학습을 대신 해주는 클래스를 정의하고 있다. verbose 여부와 관계없이 epoch당 출력은 나오도록 수정했다. 총 데이터 개수를 미니 배치 크기로 나눈 값에 입력 받은 에포크를 곱한 값을 max\_iteration으로 정의한다.

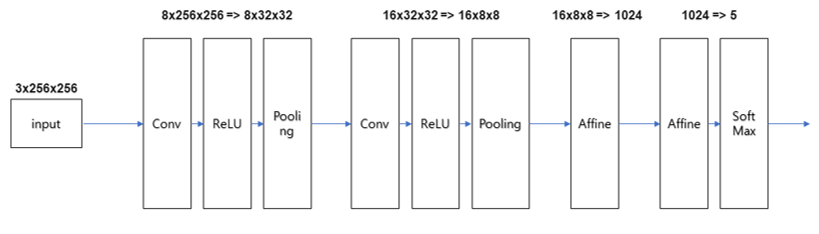
optimizer.py 파일은 최적화 클래스들을 제공한다. layers.py 파일은 계층을 구성하는 클래스들을 제공한다.

6. 학습 과정에 대한 설명

6.1. 레이어 하이퍼 파라미터 변동에 따른 실험과 결론



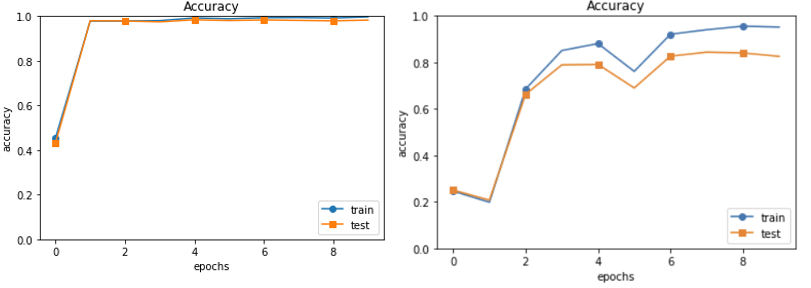
<그림 3. 중간발표 모델 1>



<그림 4. 중간발표 모델 2>

중간발표때 우리 조는 첫번째로 합성곱 레이어가 하나로 구성된 모델과 두개로 구성된 모델의 정확도를 비교했다. 또한 두번째로 합성곱 레이어의 필터의 개수, 크기, 스트라이드와 같은 레이어 하이퍼 파라미터를 조절하며 정확도를 비교했다.

그림 3은 합성곱 레이어가 하나로 구성된 모델이고, 그림 4는 합성곱 레이어가 두개로 구성된 모델이다.



<그림 5. 중간발표 모델 1과 모델 2의 정확도 비교. 좌: 모델 1, 우: 모델 2>

그림 5은 중간발표 두 모델의 비교를 나타낸다. 모델 1의 정확도에 비해 모델 2의 정확도가 확연히 떨어졌는데, 우리 조는 그 이유로 receptive field 크기의 감소가 정확도의 하락으로 이어졌다고 파악했다.

모델 1의 합성곱 레이어의 receptive field 크기는 750(30\*5\*5)임에 반해, 모델 2의 첫번째 합성곱 레이어의 receptive field 크기는 128(8\*4\*4)으로 약 1/6로 줄어든 것이다.

그러나 모델 1의 합성곱 레이어의 파라미터 숫자는 2280개(30\*3\*5\*5+30)임에 반해, 모델 2의 첫번째 합성곱 레이어의 파라미터 숫자는 392개(8\*4\*4\*3+8)로 확연히 감소했으므로, 학습 성능에 있어서는 약간의 개선을 가져왔다.

따라서 새로운 모델의 레이어 하이퍼 파라미터를 결정할 때, 한 계층의 필터의 개수가 너무 큰 경우 학습이 너무 포괄적이라 제대로 이루어지지 않거나, 파라미터 개수가 너무 많아 성능이 떨어질 것을 우려했다.

테이블이(가) 표시된 사진

자동 생성된 설명

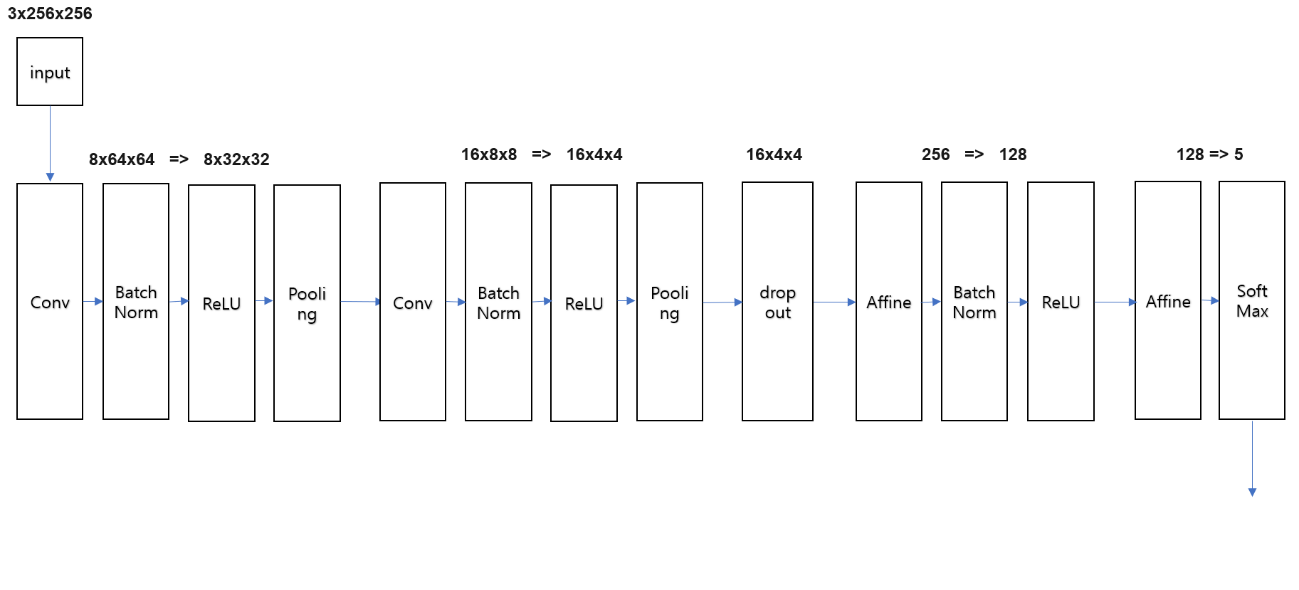
<그림 6. 중간발표 모델의 기존 정확도 출력(좌)와

레이어 하이퍼 파라미터를 2배로 한 이후의 정확도 출력(우)>

단순히 필터의 크기와 개수를 2배로 늘려 모델을 학습시킨 경우 정확도를 떨어뜨리는 결과를 얻었다. 그림 6은 필터의 개수, 크기, 스트라이드 값을 단순히 2배로 늘린 모델(우)과 그 이전 모델(좌)의 정확도의 출력이다.

따라서 상당히 높은 정확도를 얻은 중간발표 모델 1 수준의 노드의 receptive field 크기 수준으로 새로운 CNN 모델의 전체 receptive field 크기를 유지하되, 필터의 크기는 유지한 채로 계층의 개수를 늘려 각 계층의 줄어든 receptive field 크기를 보완하자는 결론을 내렸다.

6.2. 최종적 CNN 아키텍쳐 결정



<(재)그림 1. 구현한 CNN>

최종적으로 구현한 CNN은 그림 1과 같다. 3번 알고리즘에 대한 설명 장에서 상술한 것과 같다.

6.1.에서 내렸던 결론대로 필터의 크기는 유지하고, 필터의 개수가 늘어난 레이어를 쌓아가는 방식으로 전체적인 성능을 유지하며 정확도의 개선을 목표했다. 최종 모델의 합성곱 계층은 쌓인 두 계층간 2456개의((4\*4\*3\*8)+(4\*4\*8\*16)) 총 파라미터 개수로 기존 모델 1 이상의 정확도를 얻고자 했다.

필터의 개수를 결정함에 있어 합성곱 레이어의 필터로 인해 줄어드는 input 크기와 output크기의 차이를 보완하고자 했다. 256 크기의 input이 4\*4 필터를 거치며 64\*64가 되므로, 필터의 개수를 8개를 줘 일부 보완한다는 생각이었다.

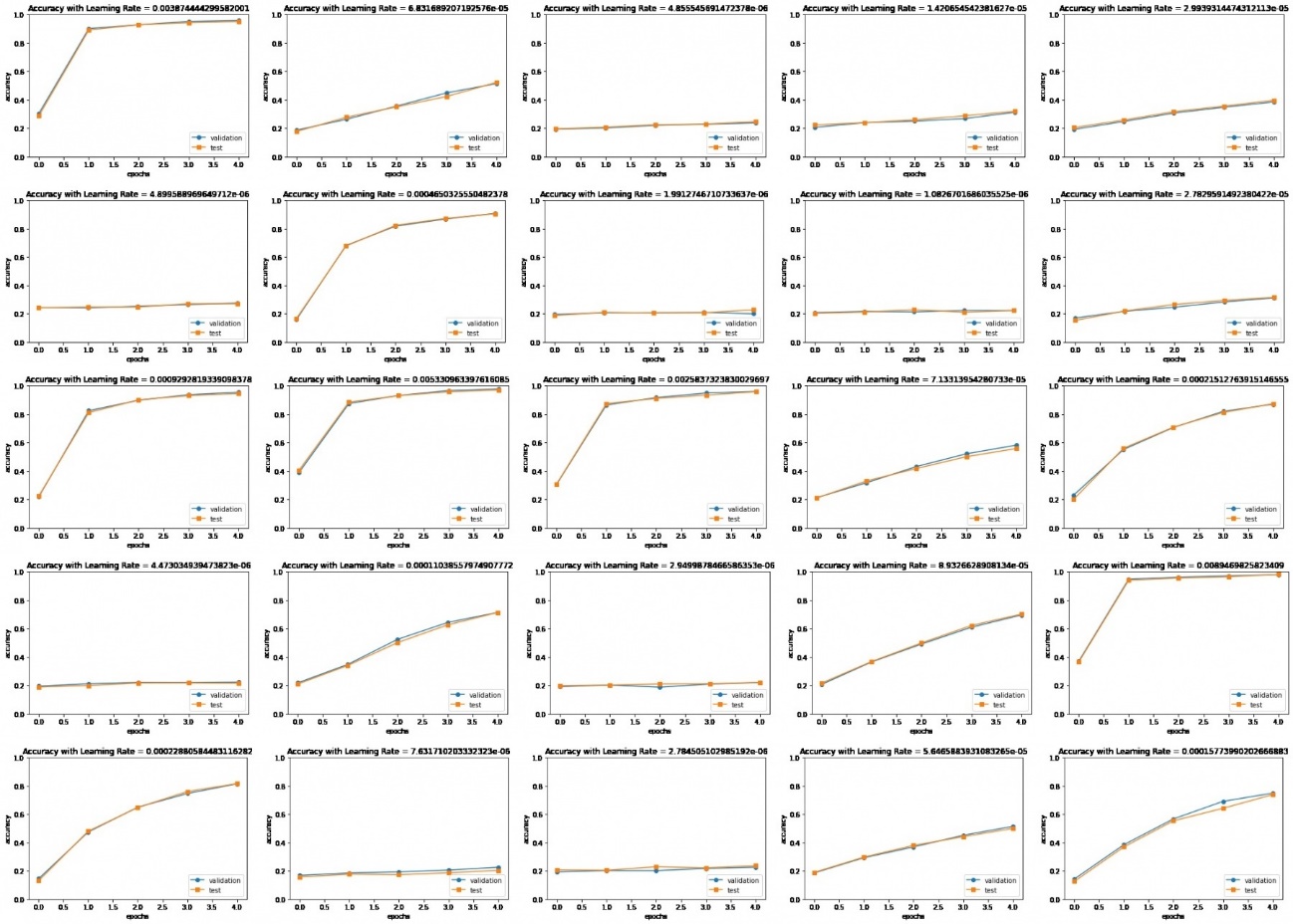
각 계층의 가중치 초기값은 ReLU에 맞춰 He 초기값을 사용했고, 또한 배치 노멀라이제이션을 적용해 가중치 소실 문제를 줄여 오버피팅 문제의 개선을 목표했다. 배치의 크기가 100으로 크지 않았으므로 momentum은 0.9를 유지했다.

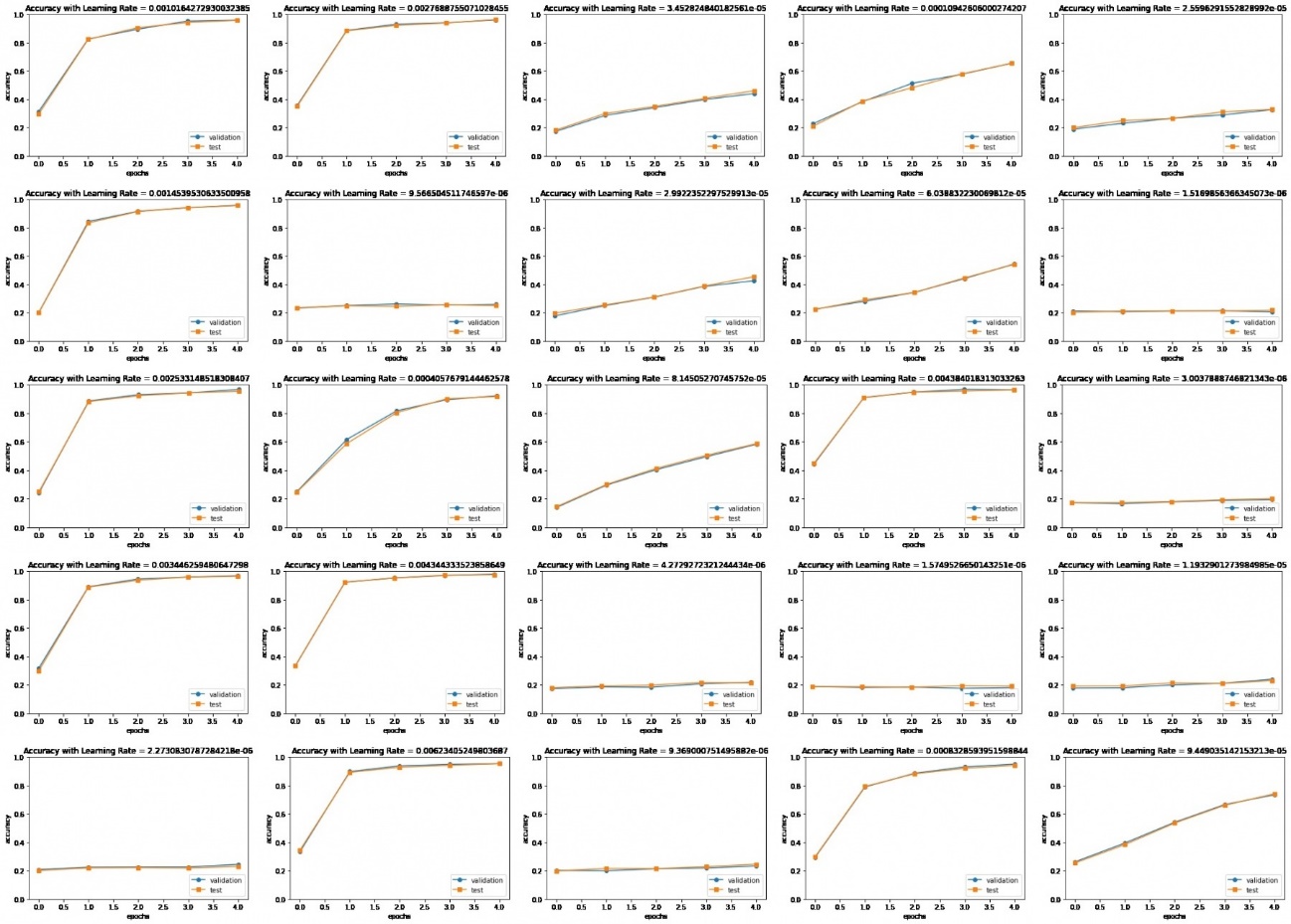
드롭 아웃을 넣어 앙상블 효과와 일부 성능 개선을 노렸다.

6.3. validation 데이터 셋을 이용한 최적 하이퍼 파라미터 결정

최적의 Learning Rate를 구하기 위해 validation 데이터 셋을 가지고 하이퍼 파라미터를 결정했다.

Epoch는 5, Learning Rate는 사이 무작위 50개 값으로 학습을 진행했다.





<그림 7. 사이 무작위 50개 Learning Rate 학습 결과 정확도 그래프>

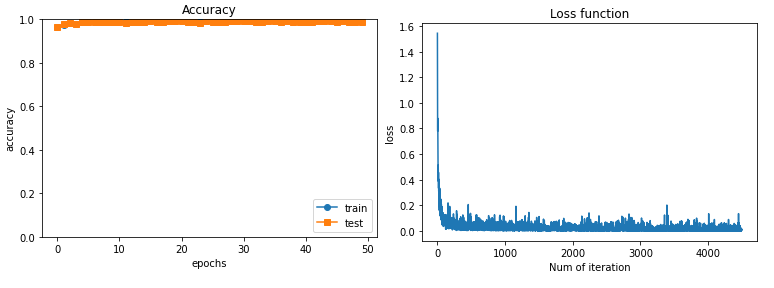
가장 높은 정확도의 결과를 얻은 Learning Rate는 ‘**0.0089469825823409**’였다.

따라서 최종 학습 시에는 Learning Rate 값으로 ‘**0.0089469825823409**’를 사용했다.

7. 결과 및 분석

최종 결정된 모델을 사용하여, epoch 50, 미니 배치 사이즈 100,

Learning Rate = 0.0089469825823409으로 학습을 진행했다.



<그림 8. 마지막 모델의 학습 결과. 정확도(좌), loss 추이(우)>

|  |
| --- |
| **<결과 출력>**  === epoch:1, current train loss:0.11239386564351628, train acc:0.9655555555555555, test acc:0.9646666666666667 ===  === epoch:2, current train loss:0.01983911275272851, train acc:0.9758888888888889, test acc:0.9776666666666667 ===  === epoch:3, current train loss:0.05178672533001546, train acc:0.9841111111111112, test acc:0.9826666666666667 ===  ...  === epoch:48, current train loss:0.0002631021525881834, train acc:0.9945555555555555, test acc:0.989 ===  === epoch:49, current train loss:0.018597494202040713, train acc:0.9953333333333333, test acc:0.9896666666666667 ===  === epoch:50, current train loss:0.01441734173897391, train acc:0.9943333333333333, test acc:0.9883333333333333 ===  =============== Final Test Accuracy ===============  test acc:0.9866666666666667 |

그림 8은 epoch 50, 미니 배치 사이즈 100, Learning Rate = 0.0089469825823409으로 진행한 학습의 결과를 나타낸다. 왼쪽 그래프는 training data와 test data에 대한 정확도, 오른쪽 그래프는 학습 중 loss값의 추이를 나타낸다.

He 초기값, 배치 노멀라이제이션, 드롭 아웃 등을 적용함으로써 training data와 test data 사이의 정확도의 차이가 0.006 가량으로 많이 낮아, 오버피팅을 상당히 해결한 좋은 결과를 얻어냈다고 보인다.

하이퍼 파라미터 최적화와 Adam의 적용으로 학습이 빠르게 높은 정확도를 향해 진행되었음을 그림 8의 그래프에서 확인할 수 있다.

그러나 여전히 loss값이 튀는 현상이 발생하는 것을 볼 수 있었는데, 최적 하이퍼 파라미터를 validation data로 결정했음에도 불구하고, 당장 epoch 5에서 가장 높은 정확도를 도출하지 못했더라도, 크기가 작은 Learning Rate를 선택해보는 시도가 필요했던 것 같다.

이 학습은 외국의 쌀 품종 데이터를 기반으로 진행되었지만, 이번에 학습된 모델을 가지고 전이 학습을 통해 국내 쌀 품종의 분류에도 역시 적용할 수 있을 것 같다.

8. 실행 메뉴얼

데이터는 캐글 ‘MURAT KOKLU’가 올린 Rice Image Dataset라는 게시글에서 받을 수 있다. 아래의 링크가 해당 게시물을 가리키고 있다.

(<https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset>)

학습을 위해서 numpy, matplotlib, scikit-learn, tensorflow 패키지를 설치해야 한다.

pickle, bz2, glob, collections은 파이썬의 표준 범용 라이브러리이지만, 없다면 이것도 설치해야 한다.

sklearn에서는 sklearn.model\_selection의 train\_test\_split()함수,

tensorflow에서는 tensorflow.keras.preprocessing.image.load\_img()함수 밖에 사용하지 않으므로 해당 함수들만 별도로 설치할 수 있다면 그 방법도 가능하다.

* **첨부 파일에 동봉된 이미 만들어진 피클 data.pickle과 labels.pickle을 사용하는 경우 3으로 건너뛰어야 한다.**

8.1. pickle\_maker.py 구동을 위해서는 numpy, tensorflow 두 패키지 설치가 필요하다.

8.2. 캐글에서 다운로드 받은 data가 들어있는 Rice\_Image\_Dataset 폴더를 pickle\_maker.py의 바로 상위 경로에 두고 pickle\_maker.py를 실행한다.

(다만 이미지 데이터셋 75,000개를 모두 사용하여 만든 피클은 train\_convnet-2.py에서 불러올 때 메모리 오류가 나는 경우가 있다.)

우리 조에서는 pickle\_maker.py의 실행 시 IDE는 PyCharm으로 진행했지만 다른 종류여도 문제 없다. 별도의 경로를 원하는 경우 코드의 20행부터 24행까지 glob.glob('**../**Rice\_Image\_Dataset/Arborio/\*.\*') 부분에서 ../ 부분을 수정하면 된다.

8.3. 생성된 피클(혹은 동봉된 피클) data.pickle과 labels.pickle과 최종 코드 layers.py, optimizer.py, our\_cnn.py, trainer.py, train\_convnet-2.py는 같은 폴더 내에 존재해야 한다.

8.4. train\_convnet-2.py을 실행한다. 모든 학습은 train\_convnet-2.py에서 이루어진다.

8.5. 41, 42행 train\_test\_split() 함수 매개변수의 random\_state를 바꿈으로써 데이터를 shuffle하는 난수의 seed를 바꿀 수 있다.

데이터를 불러오는 과정에서 메모리 초과 오류가 발생한 경우, 2번으로 돌아가 Rice\_Image\_Dataset 폴더 내 각 클래스별 이미지 파일 개수를 조절하거나, 이미 15,000개로 조절되어 만들어진 제출물에 동봉된 피클 파일을 경로상에 가져와 사용한다.

8.6. 우리 조에서 학습한 파라미터 모델을 불러와 이어서 학습을 진행하고 싶은 경우 122행 # network.load\_params(“params.pkl”) 앞 주석처리와 공백을 지우고, 제출물에 동봉된 params.pkl 파일을 앞서 피클과 코드 파일이 있는 폴더와 같은 경로에 옮긴 후, 실행하면 된다.

에포크 숫자를 바꾸는 경우 116행의 max\_epochs의 값(2로 설정되어 있다)을 조절한 후, 이에 맞춰 141, 142행의 trainer.train\_acc\_list[0:2], trainer.test\_acc\_list[0:2] 리스트도 슬라이싱 크기를 조절해줘야 한다.

125행, 126행의 Trainer class 생성자 매개변수 mini\_batch\_size=와 ‘lr’: 뒤의 값을 조절함으로써 미니 배치의 크기와 러닝 레이트와 같은 하이퍼 파라미터를 바꿀 수 있다.

8.7. 82행~113행 사이의 주석처리를 지우면 10^-6~10^-2 사이 무작위 learning rate값 50개에 대한 validation data를 통한 하이퍼 파라미터 옵티마이제이션이 이루어진다. 84행 uniform() 함수 안 매개변수 low, high, size를 조정해 learning rate의 범위와 개수를 지정할 수 있다.