인공지능 과제 리포트

과제 제목: CNN으로 과일 분류하기

학번: B811092
이름: 손현수

1. 과제 개요

본 프로젝트는 넘파이 연산관련 함수와 KERAS 라이브러리 함수를 활용해 CNN 알고리즘을 설계하여 8000개의 104*128픽셀 크기 이미지들에 담긴 각 과일의 특징을 추출하고, 이를 통해 2000개 사진에 대해 과일을 예측하는 CNN 모델을 직접 구현하는 것이다.

2. 구현 환경

Window 64bit 운영체제 위에서, VisualStudioCode Interpreter를 사용하여 과제를 구현하였다.

3. 알고리즘에 대한 설명

CNN 모델은 이미지인식의 대표적인 딥러닝모델이며 비교적 새로운 융통성 있는 Neural Network 구조이다. CNN에서 가장 핵심적인 레이어는 컨볼루션 레이어인데, 이는 입력으로 받은 사진에 대해 특징을 추출하는 레이어이다. 특징을 추출하는 과정에서는 Filter라는 개념이 사용되는데 이는 통상 3X3*, 혹은 5X5* 픽셀 크기의 패턴으로, 각 픽셀에 할당된 숫자와 필터에 포개지는 이미지 픽셀 숫자들을 곱한 후 더한 다음, 지정된 stride값만큼 오른쪽으로 이동하는 과정을 이미지 내에 모든 픽셀들을 거칠때까지 반복한다. 이러한 절차를 거쳐 추출된 결과는 특성맵에 기록되며 다음 레이어로 넘어간다. 이 특성맵은 풀링레이어를 거치면서 사이즈가 줄어드는데, 이때 자주쓰는 기법은 맥스풀링 기법이다. 이렇게 컨볼루션-풀링 레이어를 거치게 되면, 고차원적인 패턴이 담긴 특성맵이 반환되는데, 이 사이클이 계속 중첩될수록, 즉 계층이 올라갈수록 단순했던 패턴들은 점차 복잡한 이미지로 추상화(abstraction)가 진행된다.
일정 횟수 이상 컨볼루션-풀링 사이클을 거친 후에는, 최종적으로 어파인 레이어를 통과시켜, 특성맵들을 1차원 어레이로 변환 +신경망을 완전연결하여 최종 반환된 특성맵을 사진 클래스로 최종분류시킨다.

4. 데이터에 대한 설명

- Input Feature

우리 dataset은 10개 과일에 대해 1개 과일당 1000장씩 총 10000장 존재하며, 각 이미지는 (128, 104, 3)의 shape를 가지고 있다. 이는 우리의 dataset에 존재하는 이미지가, RGB라는 3개 채널을 가지는 컬러 이미지이며, 폭 128, 높이 104임을 의미한다.

10000개의 이미지 중, 8000개(각 과일당 800개)는 train data, 2000개(각 과일 당 200개)는 test data로 활용하였다

- Target Output

Target output은 10개이며,

fruit=['apple','banana','blueberry','cherry','coconut','grape','kiwi','lemon','pear','tomato']

형태이다.

5. 소스코드에 대한 설명

- main.py

일단 과일 이미지들을 담고 있는 파일에서 데이터를 가져옵니다. 이때 정렬된 데이터셋이 학습에 영향을 끼칠 수 있으므로 데이터셋을 섞어줍니다.

신경망 훈련을 위해 epoch 값, 배치 사이즈, 최적화 기법, 학습률 등을 설정합니다. CNN, 이미지 데이터, 그리고 설정한 값들로 훈련 클래스를 통해 학습합니다. 학습이 완료되면 최종적

으로 갱신한 Convolution 계층과 Affine 계층의 파라미터들을 pkl 파일로 저장합니다.

- convnet.py

작성한 CNN의 구조는 아래와 같습니다.
CNN : Convolution - Relu - Pooling - Convolution - Relu - Affine - Softmax

CNN의 Convolution 계층과 Affine 계층의 가중치를 초기화합니다. 이때 Convoulution 계층과 Pooling 계층을 통과하면서 예상되는 Affine 계층 직전의 노드의 수를 직접 계산하여 Flatten을 실현합니다. 이후 predict 메소드를 통해 정답을 예측하고, loss 메소드를 통해 예측값과 실제 정답값 간의 차이를 계산하고, accuracy 메소드를 통해 정확도를 계산하고, 그리고 gradient 메소드를 통해 기울기를 산출합니다.

총 2개의 Convolution 계층과 1개의 Pooling 계층을 통해 이미지의 특징들을 추출합니다. 추출한 특징들을 Affine 계층과 Softmax 계층을 통해 결과로 출력합니다. 활성화 함수를 위해 Relu 계층을 사용했습니다. layers.py를 통해 모든 계층을 클래스로 구현하고, forward와 backward 메소드를 작성했습니다.

- trainer.py

신경망 훈련 클래스는 네트워크, 데이터셋, 하이퍼 파라미터를 받아서 훈련망을 생성합니다.

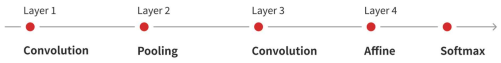
학습이 시작되면 각 iter마다 train 데이터에서 batch size만큼의 데이터를 임의로 추출합니다. 이 데이터를 통해 기울기를 산출하여 Adam 최적화 기법으로 network의 파라미터를 갱신합니다. 그다음 다시 한번 이 데이터를 이용하여 train loss 값을 구하고 출력합니다.

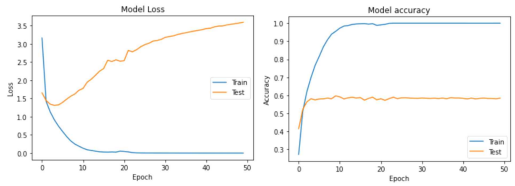
이 과정이 반복되다 1 epoch를 수행하면 정해진 수만큼의 train과 test의 샘플 데이터를 통해 정확도를 구하고 출력합니다.

모든 과정이 끝나면, 즉 정해진 epoch를 달성하면 마지막으로 test 데이터를 통해 정확도를 출력합니다.

6. 학습과정에 대한 설명

1단계



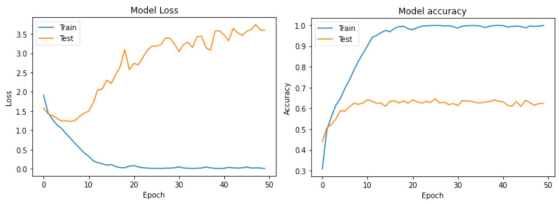


총 레이어: **4층** Loss: 3.590419292449951, Acc: 0.5839999914169312

정확도가 낮은 원인을 파악하고자 Kaggle, 등 다수의 CNN 사례에 대해 리서치를 해본 결과, 절대적인 레이어 층이 얕아 학습이 충분히 되지 않은 상태임을 파악하였음

→풀링을 진행하면서 이미지의 사이즈가 줄어들기 이전에 컨볼루션 단계를 중첩하여 레이어 층을 늘리기로 계획함

2단계



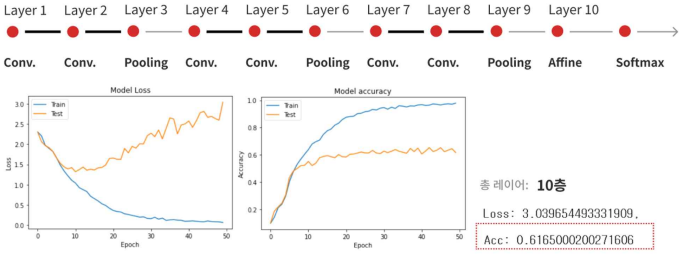
총 레이어: **7층** Loss: 3.5923757553100586, Acc: 0.6234999895095825

Pooling을 거치기 전에 Convolution Layer을 한번만 거쳤던 기존과 달리 Convolution layer을 두번 중첩하여 총 7층을 구성하였음

정확도 개선은 있었으나 우리가 목표로 했던 정확도에 미치지 못하였으며, 레이어 층 또한 리서치했던 알고리즘에 비해 아직 얕다고 분석하였음.

→데이터셋 이미지 사이즈 자체를 104128에서 208256으로 교체하였으나, 학습하는 데에 시간이 많이 소요되었음. 이에 기존 사이즈를 유지하되, Convolution에 전달되는 stride 인자값을 1로 조정하여 컨볼루션을 거쳐도 원래 사이즈를 유지할 수 있도록 하였음

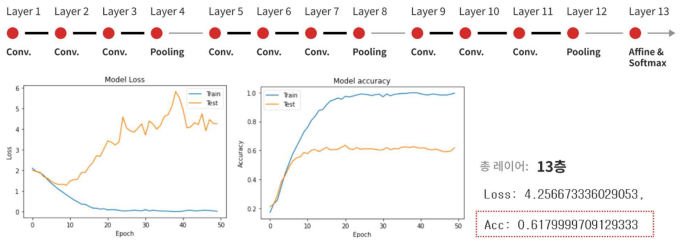
3단계



총 레이어: **10층**

Loss: 3.039654493331909,

Acc: 0.6165000200271606



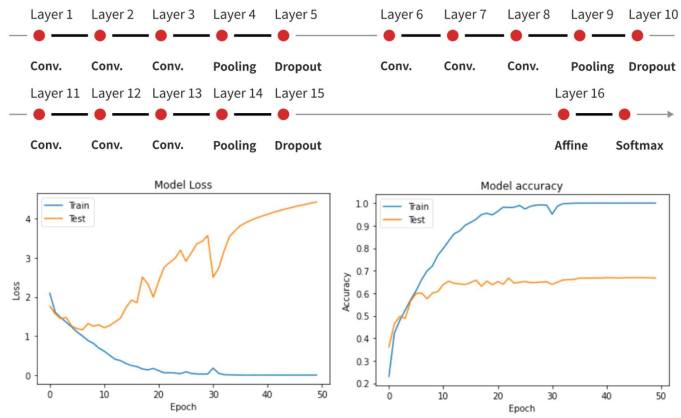
총 레이어: **13층**

Loss: 4.256673336029053,

Acc: 0.6179999709129333

1. Convolution - Pooling 사이클을 기존 2번에서 3번으로 늘림
2. 매 사이클에서의 Convolution 레이어 개수를 2에서 3으로 늘림
층을 누적시켰음에도 정확도에서의 큰 차이를 찾아볼 수 없었음. 오히려, Epoch가 진행됨에 따라 test Accurancy가 떨어졌으며, 이에 대한 원인으로 Overfitting을 꼽음
→Regularization 기법 중 하나인 Dropout 레이어를 삽입하여 이를 해결하고자 하였음

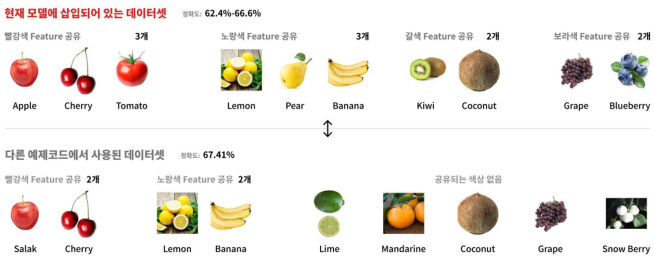
4단계



총 레이어: **16층** Loss: 4.42233419418335, Acc: 0.6669999957084656

매 사이클의 끝에 Dropout 레이어를 추가하였음
Regularization 기법을 적용 한 후 정확도가 유의미하게 상승하였음. 허나 여러번 시도하였음에도목표했던 67.41%를 근소하게 넘기지 못했음.
→참고하였던 예제 코드와 다른 과일 데이터셋을 사용하였기에, 목표했던 정확도에 미치지 못하는 원인이 데이터셋일 것이라는 가설을 세움

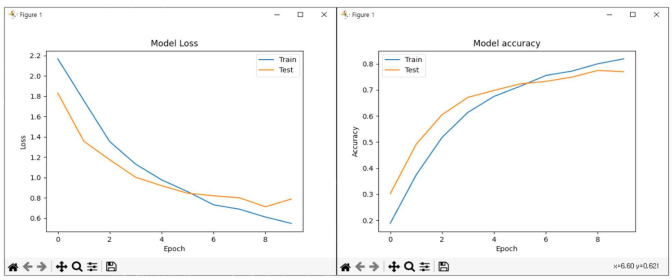
5단계



현재 모델이 사용하는 데이터셋의 경우 같은 색상을 공유하는 과일이 적어도 2개씩 있는 데에 반해, 정확도가 더 높은 예제코드의 경우 5개의 과일이 모두 고유한 색상을 지니고 있음

→데이터셋의 차이가 두 모델 간의 정확도 차이가 발생하는 주 원인으로 판단하였음

6단계



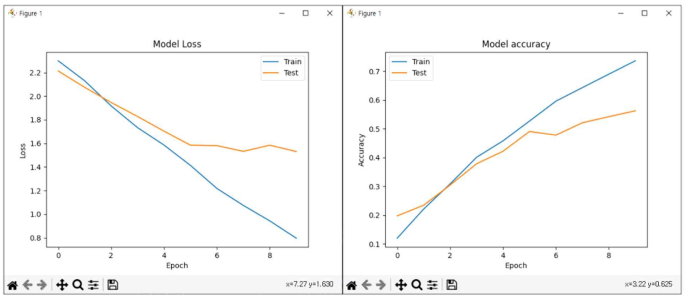

```
Epoch 8/10
40/40 [==============================] - 137s 3s/step - loss: 0.7328 - accuracy: 0.7552 - val_loss: 0.8212 - val_accuracy: 0.7325
Epoch 8/10
40/40 [==============================] - 137s 3s/step - loss: 0.6895 - accuracy: 0.7720 - val_loss: 0.8016 - val_accuracy: 0.7490
Epoch 9/10
40/40 [==============================] - 137s 3s/step - loss: 0.6126 - accuracy: 0.8000 - val_loss: 0.7130 - val_accuracy: 0.7745
Epoch 10/10
40/40 [==============================] - 137s 3s/step - loss: 0.5499 - accuracy: 0.8192 - val_loss: 0.7891 - val_accuracy: 0.7700
63/63 - 7s - loss: 0.7891 - accuracy: 0.7700 - 7s/epoch - 105ms/step

Loss: 0.7890963554382324, Acc: 0.7699999809265137
```

데이터를 분류하는 과정에서 색상 Feature가 Convolution 결과에 미치는 영향을 파악하고자, 최대한 이미지 픽셀의 색상이 겹치지 않는 과일 위주로 데이터셋을 재구성하여 학습

정확도 76.99%로 대폭 향상

→사용하는 색상이 다소 겹치는 저번 데이터셋 구성과 달리 정확도가 큰 폭으로 증가함을 확인할 수 있었으며, Test 데이터 간의 overfitting 문제 또한 해결되었음

7단계



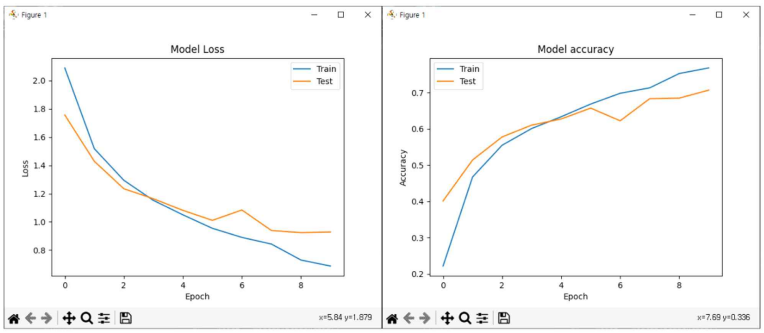

```
Epoch 8/10
40/40 [==============================] - 136s 3s/step - loss: 1.0739 - accuracy: 0.6430 - val_loss: 1.5321 - val_accuracy: 0.5210
Epoch 9/10
40/40 [==============================] - 138s 3s/step - loss: 0.9428 - accuracy: 0.6898 - val_loss: 1.5842 - val_accuracy: 0.5420
Epoch 10/10
40/40 [==============================] - 136s 3s/step - loss: 0.7966 - accuracy: 0.7361 - val_loss: 1.5305 - val_accuracy: 0.5625
63/63 - 7s - loss: 1.5305 - accuracy: 0.5625 - 7s/epoch - 108ms/step

Loss: 1.530456185340881, Acc: 0.5625
```

빨강색을 띄는 과일들 만으로 데이터셋을 재구성하여, 색상 Feature가 Convolution 과정 간 분류에 핵심적인 기준이 된다는 것을 증명하고자 하였음

정확도 56.25%로 대폭 하락

→예상했던 대로, 다른 색상들로 구성한 데이터셋 대비 훨씬 낮은 정확도를 보여, Convolution 과정 간 색상 Feature의 중요성이 증명되었음. 이와 같은 맥락으로 저번 구성에서 보이지 않았던 Overfitting 현상도 loss값의 변화를 통해 확인해볼 수 있었음

8단계




```
Epoch 8/10
40/40 [==============================] - 141s 4s/step - loss: 0.8430 - accuracy: 0.7128 - val_loss: 0.9389 - val_accuracy: 0.6830
Epoch 9/10
40/40 [==============================] - 141s 4s/step - loss: 0.7299 - accuracy: 0.7523 - val_loss: 0.9239 - val_accuracy: 0.6845
Epoch 10/10
40/40 [==============================] - 140s 4s/step - loss: 0.6873 - accuracy: 0.7678 - val_loss: 0.9283 - val_accuracy: 0.7065
63/63 - 7s - loss: 0.9283 - accuracy: 0.7065 - 7s/epoch - 110ms/step

Loss: 0.9282699823379517, Acc: 0.7064999938011169
```

아까 전 구성이 색상 Feature이 미치는 영향을 파악하기 위한 데이터셋이였다면, 이번에는 과일의 실루엣과 형태 Feature가 그룹핑에 주는 영향을 파악하고자 개성적인 형태를 띈 과일들로 데이터셋들을 재구성하였음

정확도 70.64% 기록

→정확도는 색상 Feature을 중심으로 데이터셋을 재구성하였을 때에 비해 낮았으나, 70대의 정확도를 보임을 통해 형태 또한 색상과 마찬가지로 과일분류에 있어 중요한 Feature라는 것을 확인할 수 있었음

7. 결과 및 분석

최종 **정확도 81.69% 기록**

각자 다른 색상의 과일을 데이터셋으로 설정 한 후, Epoch 값을 50회로 상향 조정한 결과, 81.69%를 최종으로 얻을 수 있었음

분석내용

- 머신러닝 모델을 설계하는 데에 있어 레이어의 구성 및 파라미터를 튜닝하는 과정 또한 매우 중요하지만, 초기 데이터셋을 정의하는 과정 역시 이 못지 않게 중요함을 깨달음
- Overfitting은 모델의 정확도를 높이기 위해선 피할 수 없는 문제이며, Dropout을 통해 이를 해결할 수 있음
- 컨볼루션 뉴럴 네트워크 모델의 정확도는 컨볼루션 레이어 층 개수에 따라 큰 폭으로 달라진다
- 풀링 레이어 한층에 컨볼루션 레이어 층을 3번 누적시킬 때 최적의 결과를 반환한다. (VGG16 모델 참고하였음)
- 컨볼루션 레이어 내에서 학습 진행률을 가장 크게 좌우하는 파라미터는 필터 개수이다. 해당 파라미터를 크게 설정할 경우, 학습 속도가 빨라지지만 다른 모델과 마찬가지로 overfitting이 발생할 가능성이 높아진다.

비록 순수 넘파이만을 이용하여 정확도 70%를 넘기는 초기 목표는 달성하지 못하였으나, CNN 구현을 위해 여러 넘파이 연산들을 활용해보며, 머신러닝 모델이 학습하는 매커니즘을 자세히 탐구할 수 있었음. 또한 KERAS 라이브러리를 활용하여 최적의 CNN 모델 레이어 구성, Overfitting 문제 해결방안 등 여러 지식들을 얻을 수 있었음. 머신러닝 모델의 파라미

터와 레이어 구성을 달리하는 것에서 벗어나, 데이터셋 자체를 새로 재구성한 후 학습하는 과정을 시도하여, 정확도가 일정수준에서 향상하지 못한 원인을 파악함과 동시에, 데이터셋의 중요성을 파악할 수 있었음. 다양한 케이스의 데이터셋으로 CNN학습 및 결과 분석을 진행하여, 컨볼루션 과정 간 데이터를 분류하는 매커니즘을 상세히 파악할 수 있었으며, 색상 Feature과 형태 Feature가 분류 간에 핵심기준으로 사용됨을 확인할 수 있었음.

8. 실행 메뉴얼

먼저 데이터는 kaggle의 데이터는
https://www.kaggle.com/datasets/f9472b258bbdab0dbc8cc773ad8c78a2fa1b997fa0cd88a476f184b78b93338c?resource=download&select=Resized_Fruits-262 에서 다운 받았습니다.

- numpy로 만든 모델

우선 import 명령어로 numpy 모듈과 sys, os 모듈을 설치합니다.
fruit.py에서 dataset을 load 하고 train dataset과 test dataset으로 나눠 줍니다. functions.py 파일에는 softmax, im2col, cross_entropy_error 등 여러 필요한 함수들이 존재하고, layers.py에는 Convolution, Pooling, Affine, Relu 등 학습에 필요한 layer들이 존재하고, optimizer.py에는 Adam optimizer를 구현, trainer.py에는 optimizer와 epoch, learning rate 등 학습에 필요한 값들을 설정하고 epoch마다 loss와 accuracy를 출력해 줍니다. convnet.py에는 학습에 필요한 모델의 layer들을 쌓고 gradient를 해 줍니다. 그리고 이곳에서 filter의 개수나 size를 설정해 줄 수 있습니다. 마지막으로 main.py는 main 함수 역할을 하며 학습을 진행합니다. 이곳에서 learning rate, epoch, batch size 등등을 설정해 줄 수 있습니다.

- keras로 실행한 모델

PIL 모듈에서 image를 import, numpy와 os 모듈을 import 하고, 그래프를 그릴 수 있는 matplotlib.pyplot과 keras 모듈을 import 해줍니다.
먼저 load_friut 함수에서 저희가 준비해 놓은 dataset에서 image들을 load 하고 train과 test 데이터로 나눠 줍니다. 그리고 keras의 model을 이용해 model 객체에 필요한 layer들을 쌓고, compile 후 fitting으로 결과를 확인합니다. 마지막으로 plot 함수들을 이용하여 그래프로 loss와 accuracy를 확인할 수 있습니다.

9. 프로젝트에서 기여한 부분

저는 numpy 모델의 convolution layer 구현, affine layer 구현, numpy 모델 parameter 값 조정으로 최적의 값 찾기, keras 모델 최적의 parameter 값 찾기 및 실행을 하였습니다.