|  |  |
| --- | --- |
| “직접 그린 과일 Image를  자동 채색 해주는 인공지능” | 12164724 최조훈  12181827 임세은  12183887 송진서 |

**목차**

|  |  |
| --- | --- |
| 1. **Introduction** 2. 주제명 / 주제에 대한 개요 3. What is CNN? 4. Source Code 5. **Modeling** 6. Data Augmentation 7. How to make CNN model 8. **Classification** 9. Data Augmentation 10. Softmax / One-hot encoding | 1. **Auto Coloring** 2. 영역별 색이 칠해지는 조건 3. **Analysis & Conclusion** 4. Analyze hyperparameters 5. Hyperparameter 와 loss value 의 상관관계 6. Layer 재구성을 통한 Model의 재정의 7. 아쉬운 점 |

1. **Introduction**
2. **프로젝트 주제**

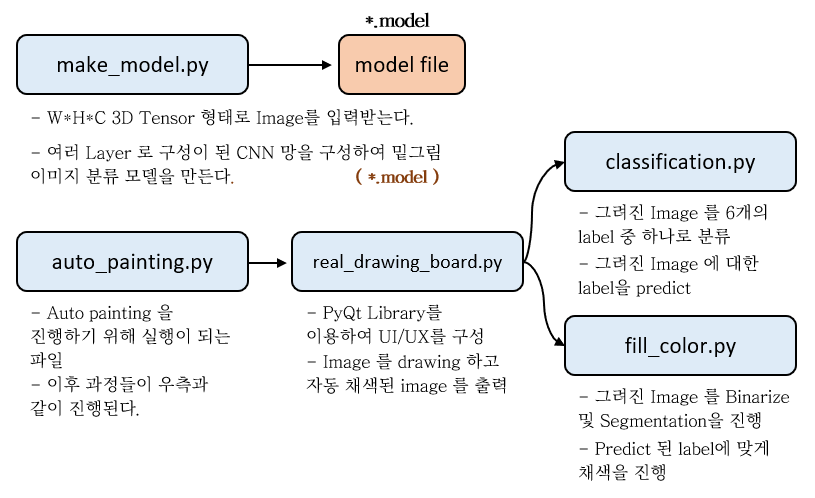
CNN을 이용하여 단색(검정색) 선으로 구성된 그려진 과일 이미지를 정해진 label에 따라 classification을 진행하고 분류한 label에 따라 과일을 자동으로 채색한다. 과일의 category는 사과, 토마토, 당근, 딸기, 수박, 참외로 지정하였다. 꼼꼼하게 그린 그림이 아닌 단색 선으로 간단하게 그린 그림이라고 할 때 사과와 토마토, 당근과 딸기, 수박과 참외는 어느정도 비슷하게 그려질 수 있다고 생각하여 이렇게 과일들을 선정하게 되었다.

1. **What is CNN?**

CNN(Convolutional Neural Network, 합성곱 신경망)은 시신경 구조를 모방해 만든 기술이다. 기존의 Fully Conneccted Layer로 구성된 Neural Network는 Data를 Vector로 처리하여 Image data의 경우 인근 Pixel과의 관계나 Pattern등 공간 정보를 고려하기 어렵다는 단점이 있었다.

그러나 CNN은 Data를 Vector화가 아닌 3D Tensor로 처리하여 공간 정보를 유지하고 Filter를 이용해 Feature map을 생성하고 학습할 수 있다. 이 때문에 Computer Vision 분야에서 우수한 성능을 보인다. 본 프로젝트는 이러한 CNN의 특성을 이용하여 직접 그린 과일 Image를 분류하고, 추가적으로 Image의 각 영역을 구분하여 자동으로 특정 색으로 채색을 진행하는 프로그램의 구현을 목표로 한다.

1. **Source Code**



1. **Modeling**
2. **Data Augmentation**

* Image resizing

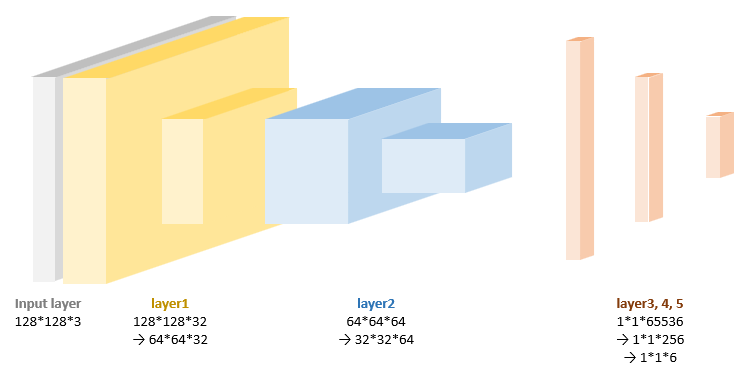
Train을 위한 image data들과 Sketched image data 들을 모두 128\*128 pixel 로 resizing을 진행하였다. 64\*64 pixel로 resizing 하는 것과 128\*128 pixel로 resizing 하는 것 중 더 세부적인 training을 위해 128\*128 pixel의 image로 resizing하는 방향으로 진행하였다.

* Image rotation

우선 [‘apple’, ‘tomato’, ‘orientalmelon’, ‘watermelon’, ‘carrot’, ‘strawberry’] 6개의 각각의 label에 대해서 직접 그린 100개의 data를 통해 train\_data를 확보하였다. 이 Image data들은 모두 하나의 (W\*H\*Channel) 형태를 가진다. 그 후 Image rotation으로 모든 Image들을 20도 단위로 rotate한 image들도 Train시에 사용하였다.

1. **How to make CNN model**

* Default CNN model



기존의 model은 위와 같이 5-layer model로 구성이 되어있다. 모든 Convolution layer 들은 3\*3 size filter와 padding을 진행하여 공간 정보를 유지하면서 channel의 개수를 2배 늘려가는 방식으로 구현되었다. 그리고 Convolution layer들을 지난 후 항상 maxPooling을 활용하여 공간 정보를 1/2로 낮추고 1/4 확률로 dropout을 진행한다. Filter 의 size 는 변경되지 않으며 channel의 깊이면 점점 2배가 되는 VGGNet 과 비슷하게 구현되었다

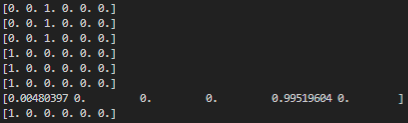
위 작업이 끝난 뒤 flatten을 시킨 후 256 length의 vector로 바꾸고 이어서 output을 도출할 수 있게끔 class의 수만큼 vector의 length를 변경한다. 이 때 마지막 Dense method에서 softmax activation function을 사용하여 각 6개의 label들 중 각각의 label일 확률 값으로 변경을 한다. 이 softmax를 통해 output된 vector는 classification.py에서 확인할 수 있다.

1. **Classification**
2. **Data Augmentation**

Modeling을 진행하는 과정에서 Resizing을 하는 Data Augmentation을 진행했으므로 동일한 조건을 위해 Sketch된 image도 294 \* 294 pixel에서 128\*128 pixel로 resizing을 진행한다.

1. 텍스트, 전자기기, 키보드이(가) 표시된 사진

   자동 생성된 설명**Softmax**



좌측 경우는 Model을 통해 예측이 된 label vector들이다. Model이 마지막에 output vector를 출력할 때 Softmax activation function을 이용하였기 때문에 각 label들에 해당되는 값들은 그 label일 확률에 해당된다. 좌측과 같이 정의가 된 output vector의 모든 element값들을 우측처럼 소수 셋째자리까지 반올림한다.

그 결과 아래에서 2번째 vector에서 첫 번째 label 의 값은 0.00480397에서 0.005가 되고, 다섯 번째 label의 값은 0.99519604에서 0.995가 되어 더 다루기 쉬운 vector의 형태를 띄는 것을 알 수 있다.

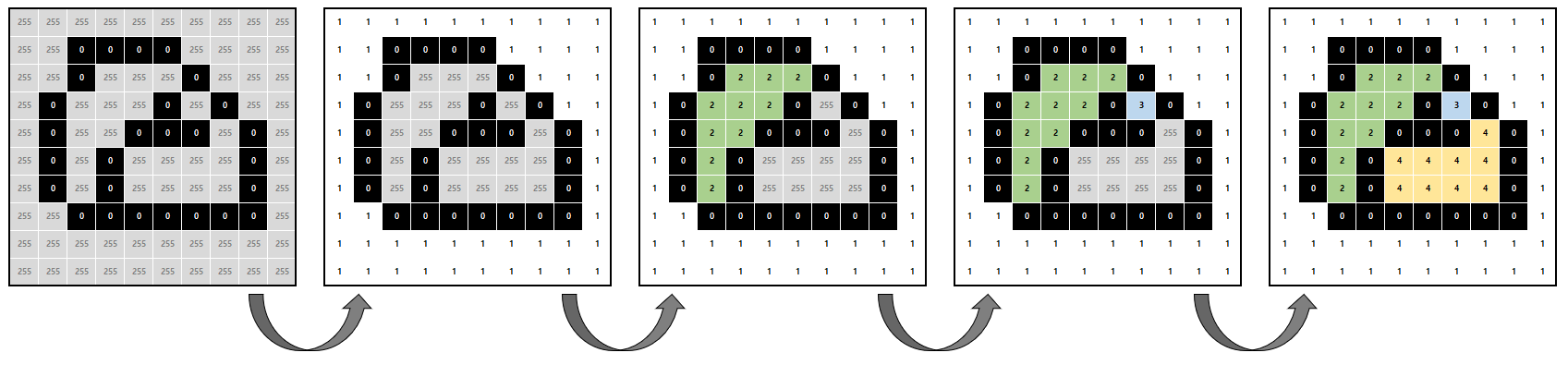
텍스트이(가) 표시된 사진

자동 생성된 설명

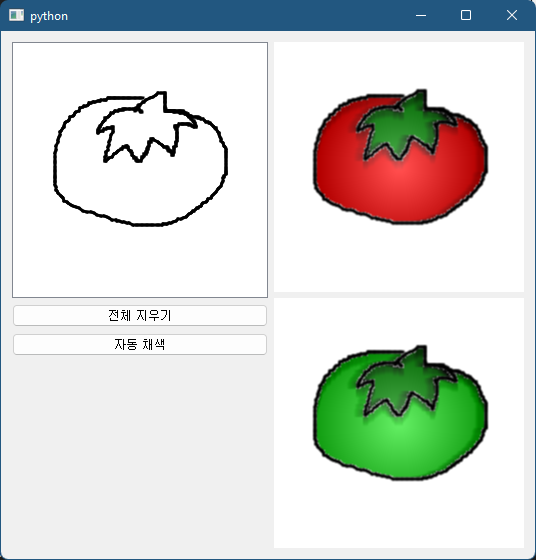
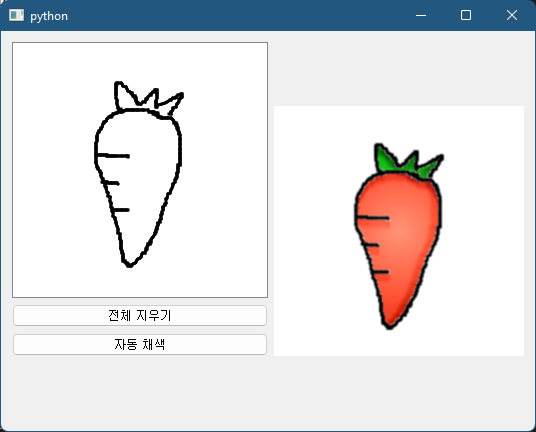
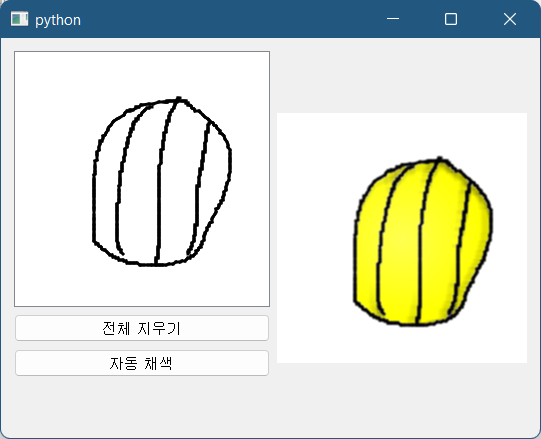
최종적으로 위 코드와 같이 label vector에서 0.8이상의 값을 띄는 label이 현재 입력 data의 predicted label이 된다. If 문의 가장 마지막 위치에서 가장 큰 확률을 가지는 label이 0.8 미만의 확률 값을 가진다면 그 label을 predicted label로 치지 않아 없는 데이터로 출력하게끔 하였다.

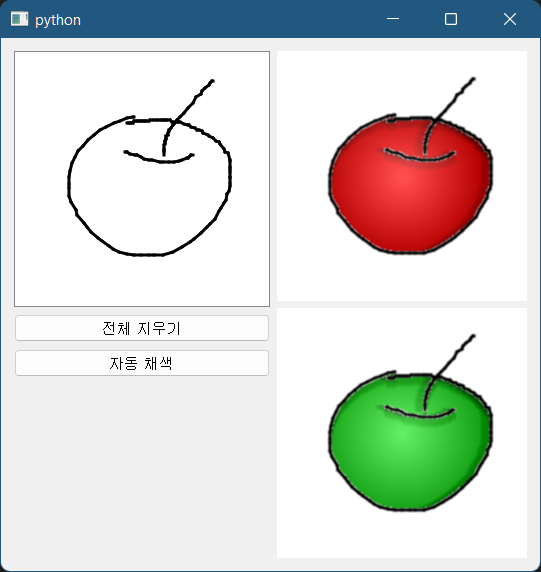
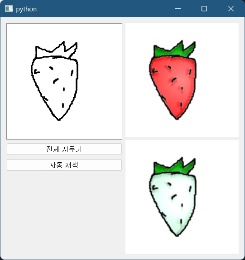
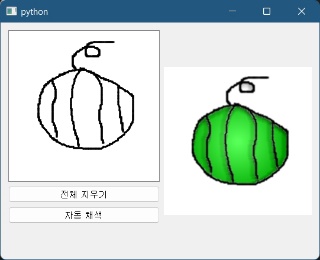
1. **Auto Coloring**

Coloring을 진행하는 과일은 두 부류로 나뉜다. 두 가지 종류의 색을 칠하는 과일과 한 가지 색만 칠하는 과일이 있다. 사과, 딸기, 토마토의 경우에는 주로 빨간색을 떠올리지만 초록색인 것도 있기에 초록색도 추가로 칠해지게끔 하였다. 이는 사용한 기존 Open source의 class및 method를 수정하여 label에 따라 유동적인 종류의 색을 칠할 수 있는 함수를 구현하였다.

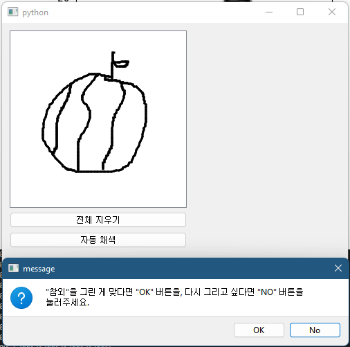


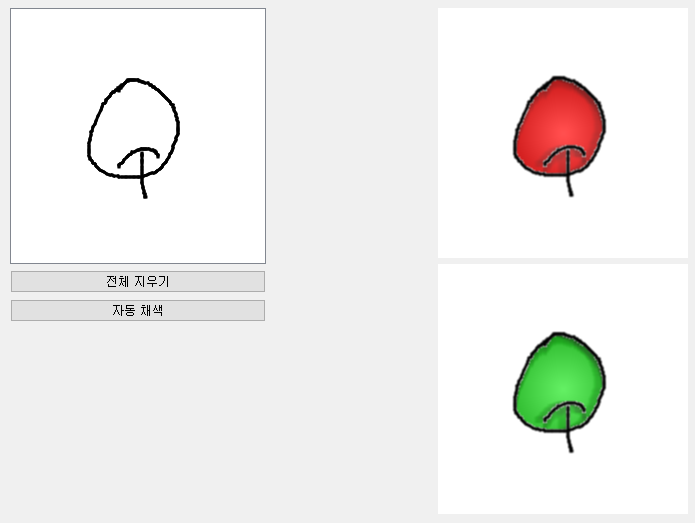
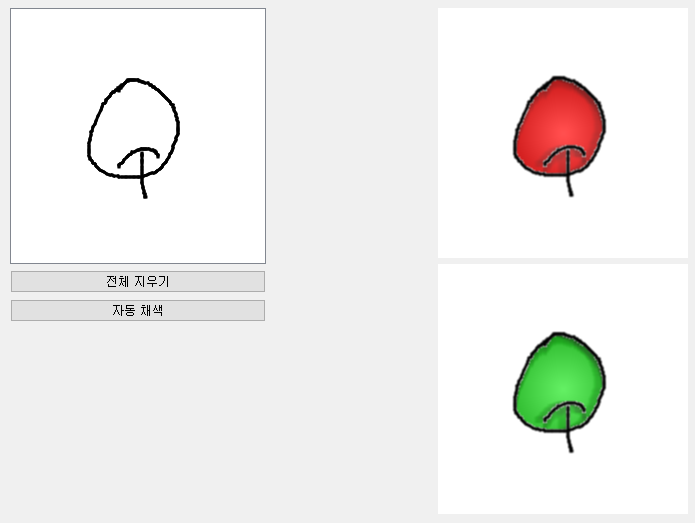
가장 먼저 binarize를 통해 선과 선이 아닌 부분을 0 또는 255로 구분한다. 이후 위 그림과 같은 방식으로 0행 0열의 pixel 부터 255의 값을 가지는 경우를 아직 색이 부여되지 않은 부분으로 보고 count값을 1부터 시작해 인접한 픽셀에 같은 count값을 부여한다. count 값에 따라 같은 색을 칠하는 방식으로 coloring이 진행된다.

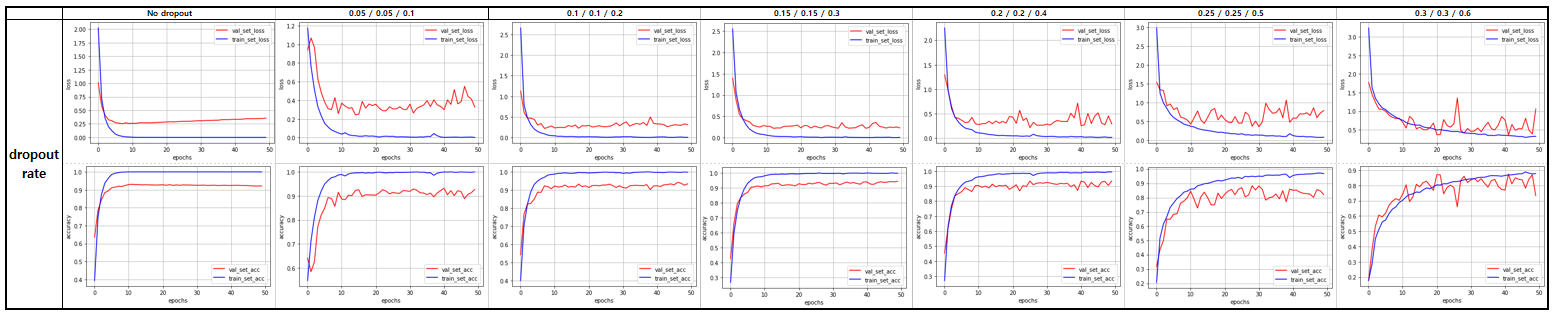
실행 결과 위와 같이 label에 따라 그 RGB code를 이용하여 그림의 내부가 채색되는 것을 볼 수 있다. 꼼꼼하게 그린 그림이 아닌 마우스로 간단하게 그림을 그린 결과, 위와 같이 사과와 토마토, 당근과 딸기, 참외와 수박이 비슷하게 sketch가 된 것을 확인할 수 있다.

또한, Training 시 회전된 그림 또한 이용했기에 좌측의 사과 그림처럼 거꾸로 뒤집힌 그림 또한 제대로 채색된다.



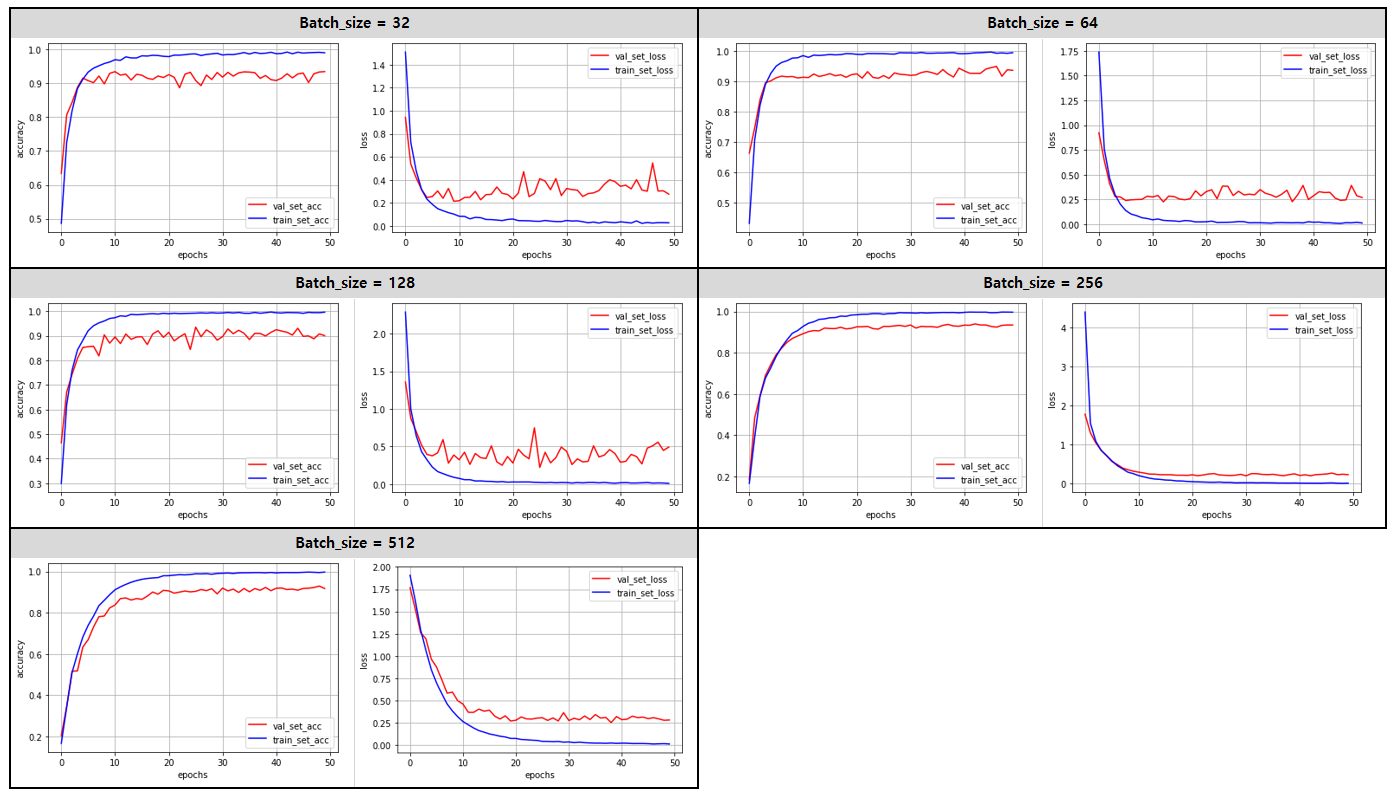
그리고 그 옆의 수박 그림과 같이 구분이 모호한 그림의 경우 제대로 classification을 수행하지 못하기도 한다.

1. **Analysis & Conclusion**
2. **Analysis : Augmented data를 활용한 model의 효율성 확인**



순서대로 dropout\_rate 가 0인 경우부터 **0.05 / 0.05 / 0.1** | **0.1 / 0.1 / 0.2** | **0.15 / 0.15 / 0.3** | … | **0.3 / 0.3 / 0.6** 인 경우에 해당된다. **2개의 convolution layer 에 앞에서 두 경우의 dropout rate가 적용**되고, **256 length를 가지는 Dense layer 에서 마지막 dropout rate를 가진다**. Dropout rate 가 **0.15 / 0.15 / 0.3** 인 경우가 가장 좋은 그래프의 형태를 보여준다.

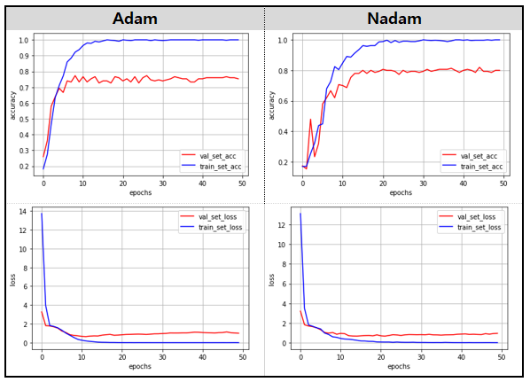
가장 좋은 성능을 내는 dropout rate 를 가지고 batch\_size 를 32 / 64 / 128 / 256 / 512 로 지정하여 실험 해 보았다.



Augmented data를 활용한 경우에는 batch size 가 256인 경우 가장 좋은 성능을 내었다. Learning rate도 탐구를 해 보았지만 0.001 이외의 값들은 비효율적인 것을 또한 확인하였다.

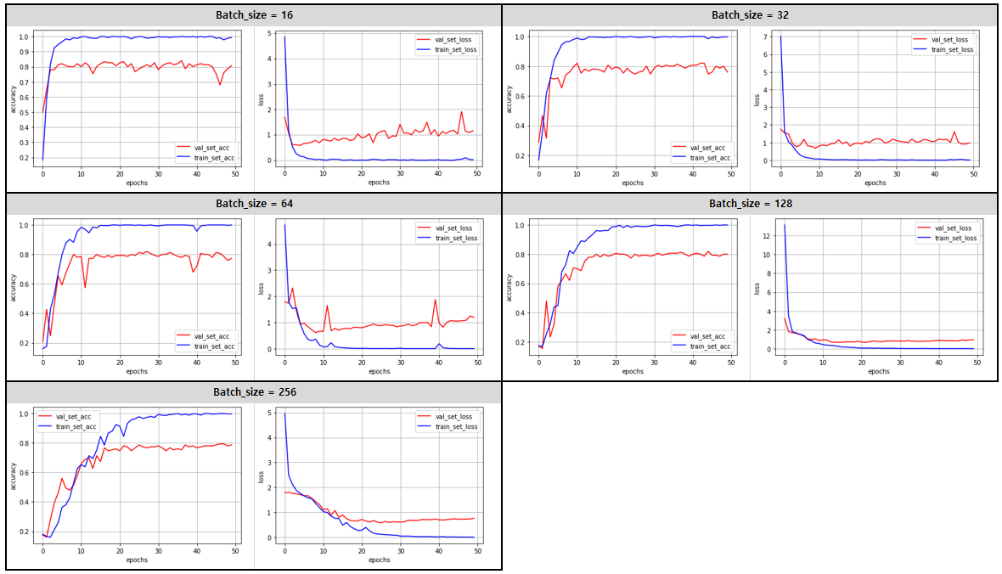
1. **Analysis : Augmented data를 활용하지 않은 model을 통한 효율성 확인**

Augmented data 를 활용하기 전에 탐구를 하였던 내용들과 Augmented data를 활용한 후에 탐구한 내용들이 상이한 것을 확인하였다. Augmented data를 활용하지 않은 model의 탐구 내용은 다음과 같다. Plain train data만 활용하여 각 Hyperparameter(optimizer, batch, epoch, learning\_rate)를 바꾸어 보면서 효율성을 비교하였다.

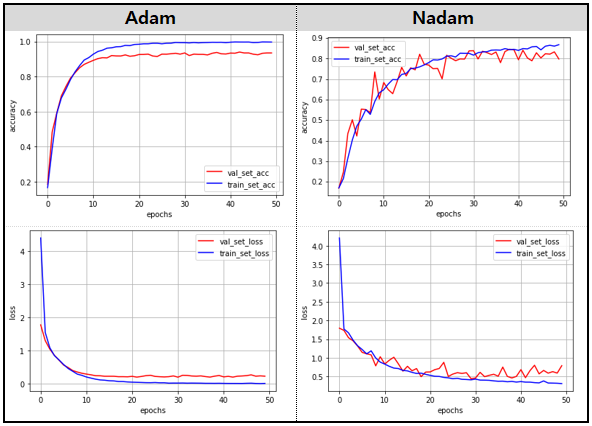
* **Optimizer 비교 (Adam, Nadam)**

기존의 Adam Optimizer를 Nadam Optimizer로 바꿔보았다. Batch size는 128로 50개의 Epoch를 실행해 본 결과 위와 같았다. 그 결과 기존의 Adam Optimizer를 사용한 것 보다 Nadam Optimizer를 사용한 경우 validation data set의 accuracy, loss 값이 Adam optimizer를 사용한 경우보다 더 성능이 좋은 것을 확인하였다. 따라서 이후 여러 Hyperparameter들을 바꿔보면서 효율성을 체크하는 과정에 Nadam optimizer를 사용하였다.

* **Batch size 비교**

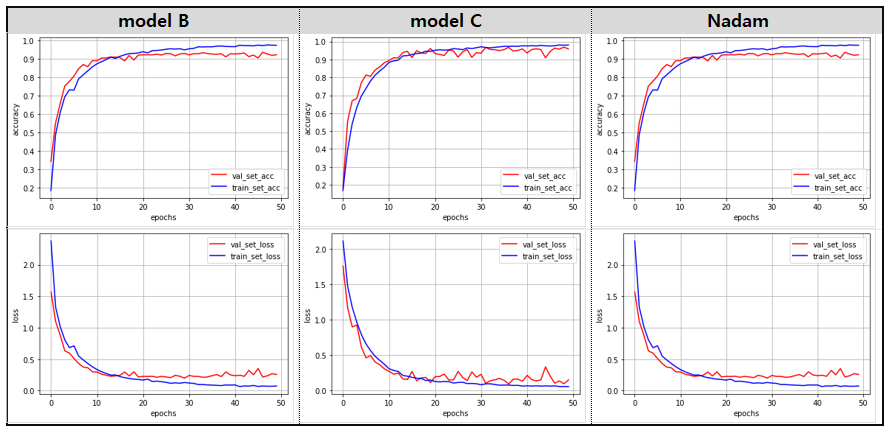
Plain test data 만 가지고 학습을 하였을 때는 Batch size 가 128인 경우 가장 좋은 성능을 내는 것을 확인하였다. Plain test data 만 가지고 학습하는 만큼 Data의 수가 적어서 이전 Augmented data를 가지고 한 경우보다 더 작은 size 에서 효율적인 모델이 구현되었다.

1. **Conclusion**
2. **Batch Size – 32, 64, 128, 256, 512** : Dataset의 크기에 따라 가장 효율적인 batch size의 값이 달랐다. Augmented data를 통한 모델링의 경우 8,100개의 Dataset이 사용되며 이 경우에는 batch size의 값이 256인 경우 가장 좋은 모델이 구현되었다. 하지만 Augmented data를 활용하지 않은 모델링의 경우에는 600개의 Dataset이 사용되며 이 경우에는 batch size의 값이 128인 경우 가장 좋은 모델이 구현되었다.
3. **Dropout rate** : dropout 없이 모델을 훈련할 경우 모델이 train data에 너무 맞춰져서 train accuracy는 1.0이 되고 validation accuracy는 epoch를 반복할수록 감소하는 overfitting이 발생한다. 전반적으로 dropout rate가 커질수록 혹은 상당히 작은 값을 가진 경우 training accuracy가 서서히 높아지며 epoch별 accuracy, loss 값의 진폭이 점점 커지는 것을 확인하였다. 실험 결과 Augmented data를 활용한 모델링에서 가장 성능이 우수한 0.15 / 0.15 / 0.3 dropout rate를 사용해 이후 실험을 진행했다.
4. **Learning rate** : learning rate가 너무 작으면 매우 느린 속도로 모델이 개선되는 모습을 보였고, 너무 크면 발산하여 모델이 생성되지 않기도 했다. 그렇기에 적절한 크기의 learning rate를 설정해야 한다. 이에 4종류의 batch size와 6종류의 learning rate를 이용해 batch size에 따른 최적의 learning rate를 탐색하자 다음과 같은 결과를 얻었다. Batch size가 커질수록 정확도가 가장 높은 최적의 learning rate 또한 커지는 경향을 보인다. 또한 learning rate에 대한 loss 및 accuracy 그래프를 그렸을 때 최적의 learning rate를 기준으로 대칭적으로 성능이 나빠지는 양상을 보였다. Batch size에 비해 learning rate가 지나치게 클 경우 학습 결과가 어느 선으로 수렴하지 못하고 발산하여 제대로 된 결과를 얻지 못하기도 한다.
5. **Optimizer** : 기존의 model은 Adam optimizer를 사용한다. Adam optimizer에 NAG를 결합한 optimizer 인 Nadam과 비교를 해 보았다.

Augmented data를 활용한 모델링 과정에선 기존의 model의 dropout rate를 0.15 / 0.15 / 0.3 으로, learning rate를 0.001로, batch size를 128로 한 경우에 가장 잘 나왔다. 이 모델을 Nadam optimizer로 변경을 하여 실행을 해 봤지만 좌측과 같이Adam optimizer을 활용한 경우보다 훨씬 좋지 않은 결과를 보였다.

하지만 Augmented data를 활용하지 않은 model은 이전 설명과 같이 Nadam optimizer를 통해서 더 좋은 model을 생성할 수 있었다. 요새 가장 많이 사용되는 optimizer가 Adam이지만 그렇다고 해서 그 optimizer에만 국한되면 안된다는 깨달음을 얻었다.

1. **Custom Model** : 기존 모델에 layer를 추가하거나 그 크기를 변경해 성능을 확인하는 실험을 진행했다. 모델 b는 기존 모델에서 128 채널의 conv layer를 추가했고, 모델 c는 모델 b에서 Flatten 이후 Dense의 크기를 256에서 128로 변경했다. 모델 d는 모델 c의 dropout rate를 마지막 제외 0.1, 마지막은 0.2로 설정했다. 실험 결과 layer 추가나 channel의 깊이 증강 등이 오히려 성능을 낮추는 요인이 되었으며 현재 data set에서는 기존의 모델이 가장 최적화된 성능을 보임을 알 수 있었다.



1. **References**

* tistory / 2021.08.25 작성 / 2022.04 접속 / [CNN을 통한 구름 분류 (tistory.com)](https://junseokk.tistory.com/4)
* github.io / 2017.05.08 작성 / 2022.05 접속 / [컨볼루션 신경망 모델 만들어보기 (tykimos.github.io)](https://tykimos.github.io/2017/03/08/CNN_Getting_Started/)
* github / 2022.02 수정 / 2022.05 접속 / [googlecreativelab/quickdraw-dataset: Documentation on how to access and use the Quick, Draw! Dataset. (github.com)](https://github.com/googlecreativelab/quickdraw-dataset#get-the-data)
* Tensorflow / 2022.01.24 수정 / 2022.05 접속 / [데이터 증강  |  TensorFlow Core](https://www.tensorflow.org/tutorials/images/data_augmentation?hl=ko)
* ko.d2l.ai / / 2022.05 접속 / [3.13. 드롭아웃(dropout) — Dive into Deep Learning documentation (d2l.ai)](https://ko.d2l.ai/chapter_deep-learning-basics/dropout.html)
* github.io / 2017.01.27 작성 / 2022.05 접속 / [컨볼루션 신경망 레이어 이야기 (tykimos.github.io)](https://tykimos.github.io/2017/01/27/CNN_Layer_Talk/)
* tistory / 2020.06.24 작성 / 2022.05 접속 / [tf.keras.callbacks.LearningRateScheduler (tistory.com)](https://dodonam.tistory.com/178)
* github.io / 2019.09.22 작성 / 2022.06 접속 / [[딥러닝] 딥러닝 최적화 알고리즘 알고 쓰자. 딥러닝 옵티마이저(optimizer) 총정리 - 재야의 숨은 초보 (hiddenbeginner.github.io)](https://hiddenbeginner.github.io/deeplearning/2019/09/22/optimization_algorithms_in_deep_learning.html#Adagrad)
* tistory / 2020.12.29 작성 / 2022.06 접속 / [[딥러닝 학습 팁] Drop out :: Cara's Moving (tistory.com)](https://pongdangstory.tistory.com/536)
* blog / 2017.11.01 작성 / 2022.06 접속 / [그림으로 보는 다양한 Optimization 기법들 (hyunw.kim)](https://hyunw.kim/blog/2017/11/01/Optimization.html)