[Project #2]

Deep Learning for

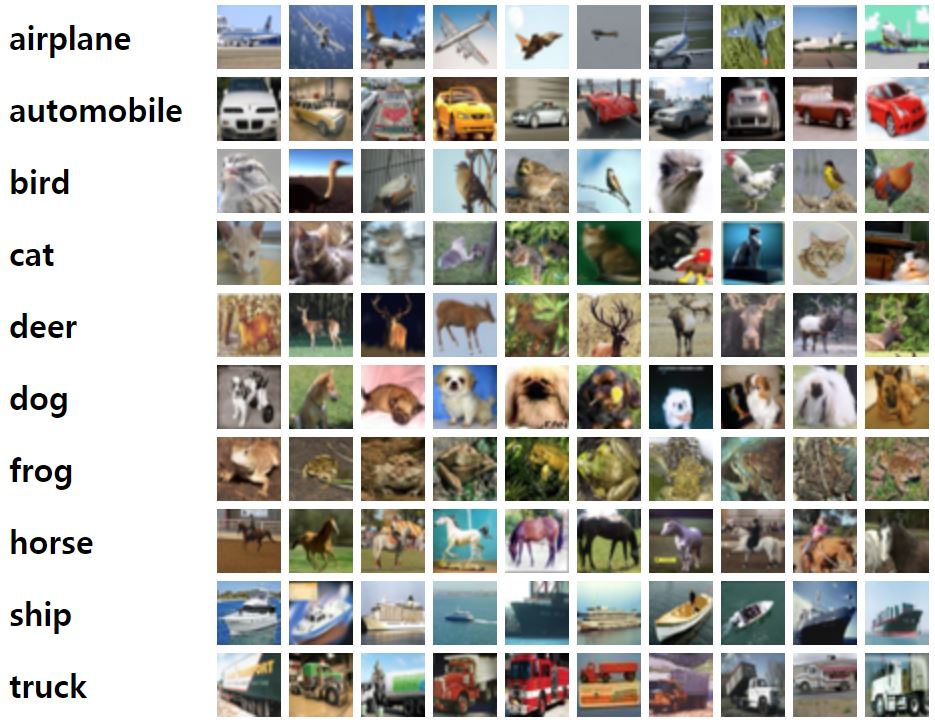
Image Classification

[HBO (#팀6)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **이름** | **학번** | **학년** | **E-mail** |
| 팀장 | 박영준 | 20170287 | 3 | qkrdudwns98@naver.com |
| 조원1 | 변지현 | 20180298 | 4 | bijh1999@naver.com |
| 조원2 | 우민식 | 20170303 | 3 | 9737638@naver.com |

# 

# **1** **Introduction**

본 프로젝트는 CIFAR-10 dataset을 분류하는 모델을 제작하고 이를 설명하기 위한 중간 과정들을 시각화하는 것을 목표로 한다. CIFAR-10 dataset은 각각의 데이터가 10개의 클래스(비행기, 자동차, 새, 고양이, 사슴, 개, 개구리, 말, 배, 트럭) 중 하나로 라벨링 되어 있는 60000개의 32\*32 RGB 이미지로 이루어져 있다.

이미지 분류 기법으로는 CNN(Convolution Neural Network)을 사용한다. 기존의 완전 연결 계층을 이용한 방법은 이미지를 1차원으로 펼치면서 이미지 데이터의 형상이 무시되는 치명적인 단점을 가진다. 하지만 Convolution layer를 사용함으로써 이미지가 가지는 형상을 그대로 유지하며 입력으로 받아들일 수 있고 출력에서도 동일하기 때문에 CNN에서는 이미지 데이터처럼 형상을 가지는 데이터를 다루기 용이하며 일반적으로 정확도 또한 높다. 데이터 시각화 및 분류되는 과정을 설명하기 위한 기법으로 Grad-CAM(Class Activation Map)과 t-sne 기법을 사용할 예정이다.

# **2** **Main Structure**

본 프로젝트는 show\_accuracy(), load\_image\_and\_label\_regularated(), make\_gradcam\_heatmap(), save\_and\_display\_gradcam(), get\_tsne() 함수들로 구성되어 있다.

show\_accuracy() 는 각각 은닉층을 지나면서 모델이 학습되면서 학습데이터에 대한 accuracy와 테스트 데이터에 대한 accuracy를 출력시키게 해주는 기능의 함수이다.

load\_image\_and\_labal\_regularated는 CIFAR 10 에 존재하는 각각의 이미지 파일과 이미지파일마다 해당하는 라벨을 불러와 50,000개의 이미지는 train\_images로 나머지 10,000개의 이미지는 test\_images로 분류하는 기능을 한다. 분류한 이미지를 32,32,3사이즈로 reshape을 해준 뒤, 255.0으로 나누어 표준화를 진행시켜주었다. 그리고 이러한 이미지들과 각 이미지에 해당하는 라벨을 train과 test로 분류하여 return시켜주는 함수이다.

make\_gradcam\_heatmap : 모델을 학습시키면서 모델이 각 이미지마다 어디를 특징으로 하는 지 확인하기 위해서 gradCam을 보여주는 기능을 하는 함수이다. 본 프로젝트에서는 colormap중 Jet를 사용하여 진행하여 집중적으로 보는 부분을 빨간색, 집중적으로 보지않는 부분을 파란색으로 출력시켜 진행하였다.

save\_and\_display\_gradcam : grad-cam을 진행한 것을 원본이미지에 합치는 과정을 하는 함수이다. 원본 grad-cam에서는 colormap이 jet로 설정되있지 않아 어느 부분을 집중적으로 보는 지 명확히 나오지 않아 이 함수 내에서 jet로 바꾸어 원본이미지와 합쳐 진행하였다.

get\_tsne : CIFAR-10에 있는 이미지 데이터를 학습하여 각 데이터들을 2차원으로 차원축소를 위해 진행한 함수이다. 먼저 차원축소를 위해 t-sNE방법을 사용하였다. tsne를 통해 차원축소 후 2차원 그래프에 각 데이터들을 띄워 시각화를 진행시켜주는 함수이다.

# 

# **3** **Training the Network**

본 프로젝트의 학습모델은 tensorflow 라이브러리를 기반으로 구현되어있으며 아래의 기법들을 사용하여 모델을 학습하였다.

최종적으로 학습시킨 모델은 아래와 같다. 본 프로젝트에서는 분류 모델의 구현에 tensorflow 라이브러리를 사용하였으며, 데이터의 입력 사이즈는 32\*32\*3, 학습 횟수(epochs)sms 100, batch\_size는 512, dropout확률은 0.5로 지정해주었다.

|  |
| --- |
| import numpy as np  import tensorflow as tf  import matplotlib.pyplot as plt  from matplotlib import cm  from tensorflow import keras  from tensorflow.keras import layers  from tensorflow.keras import regularizers  from tensorflow.keras.datasets import cifar10  from keras.utils import np\_utils  input\_shape = (32, 32, 3)  num\_classes = 10  epochs = 100  batch\_size = 512  dropout = 0.5  layers.Conv2D(64, 3, padding='same', activation='relu', kernel\_regularizer=keras.regularizers.l2(0.001)),  layers.MaxPooling2D(),  layers.Conv2D(128, 3, padding='same', activation='relu', kernel\_regularizer=keras.regularizers.l2(0.001)),  layers.MaxPooling2D(),  layers.Conv2D(256, 3, padding='same', activation='relu', kernel\_regularizer=keras.regularizers.l2(0.001)),  layers.MaxPooling2D(),  layers.Dropout(dropout),  layers.Conv2D(512, 3, padding='same', activation='relu', kernel\_regularizer=keras.regularizers.l2(0.001)),  layers.MaxPooling2D(),  layers.Dropout(dropout),  layers.Flatten(),  layers.Dense(512, activation='relu'),  layers.Dense(num\_classes,activation='softmax')  callback = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10)  model15\_2.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  result15\_2 = model15\_2.fit(  x\_train, y\_train,  batch\_size=batch\_size,  epochs=epochs,  validation\_data=(x\_test, y\_test),  callbacks=[callback]  ) |

각 과정에 대한 상세 설명은 아래와 같다.

**Conv2D (컨볼루션 필터의 수, 컨볼루션 커널의 크기, padding=(valid,same),activation=(linear,relu,sigmoid,softmax), kernel\_regularizer=keras.regularizers.l2(0.001))**

필터를 통해 입력된 이미지에 대해서 특징들을 뽑아주는 역할을 한다.

여기서 padding은 경계를 어떻게 처리할 지 정의해주는 부분이다. 우리 프로젝트 내에서는 same을 사용하여 출력 이미지 사이즈가 입력 이미지 사이즈와 동일하도록 설정해준다.

그리고 activation은 활성화 함수를 설정하는 부분이다. 활성화함수에는 linear방식, relu방식, sigmoid방법, softmax방법을 선택할 수 있다. linear방시은 디폴트 값으로 입력뉴런과 가중치로 계산한 값이 그대로 출력으로 나오게 된다. 그리고 relu방식은 우리 프로젝트에서 사용하는 방법으로 Max(x,0)의 방식으로 음수의 경우에는 0이 출력되고 양수의 경우 그 값이 그대로 출력되도록 해주는 활성화 함수이다. 그리고 sigmoid의 경우 이진분류 문제에서 출력층에 주로 사용하는 방식으로 이번 프로젝트에서는 진행하지 않았다. 그리고 softmax방식은 다중 클래스 분류문제에서 출력층에서 주로 사용되기 때문에 이번 프로젝트에서 마지막 Fully Connected Layer에서 사용하였다.

본 프로젝트에서는 과대적합을 피하는 처리과정으로 Conv2D에서 kernel Regularizer을 사용하였다. kernel Regularizer에서 L2 Regularization을 사용하여 진행하였다. 이는 가중치 행렬의 모든 원소를 제곱 하고 0.001을 곱하여 네트워크의 전체 손실에 더해진다는 의미로 사용하여 훈련할 때만 추가하여 실행하였다.

**Dropout( )**

신경망 모델이 복잡해지면서 모델이 오버피팅되는 것을 방지하기 위해서 dropout을 사용한다. dropout은 신경망 모델에서 몇개의 연결을 임의로 끊어 남은 노드들을 통해서 훈련을 하여 신경망 모델의 일반화 능력을 키워주기 위해 사용하였다.

**Pooling( )**

Pooling과정에는 max pooling과 Average Pooling 두가지 방법이 존재한다. 우리 프로젝트에서는 각 요소들의 평균 값을 뽑아내는 방법인 Average Pooling이 아닌 각 요소들의 최대값을 뽑아내는 Max Pooling방법을 사용하였다.

**MaxPooling2D( )**

pool의 사이즈를 정해주면 정해진 Pool의 크기로 잘라낸 후 가장 큰 값을 뽑아내는 방법이다. Pooling과정에서 특징을 잘 뽑아내기 위해서 가장 큰 값을 뽑아내는 Max Pooling을 선택하여 진행하였다. 이러한 Pooling 과정을 통해 이미지 사이즈를 줄이고 오버 피팅을 조절하였다. 그리고 Pooling 사이즈 중 가장 큰 값을 뽑아내어 Pooling을 했을 때 특징을 좀 더 잘 잡을 수 있도록 하였다.

**Flatten( )**

이미지를 일차원으로 바꿔주는 레이어로 컨볼루션 레이어와 맥스풀링 레이어를 반복적으로 거칠 경우 주요 특징만 추출되고 이렇게 추출된 주요 특징들은 Fully Connected 레이어( FC레이어, 전결합층)에 전달되어 학습된다. 하지만 Fully Connected 레이어에 전달되기 위해서는 1차원으로 변환해주어야 하는데 이때 사용되는 것이 Flatten레이어이다.

**Dense(출력 뉴런수, activation=(linear,relu,sigmoid,softmax))**

모델 구성 시에 dense layer를 두 번 사용하여 진행하였다. 먼저 512개를 출력시키고 활성화함수로 relu방식을 사용하는 dense함수와 마지막 부분에 클래스의 개수만큼 출력시키고 활성화함수로 softmax방식을 사용하는 dense를 두가지를 사용하여 진행했다. softmax방식은 입력받은 값을 출력으로 0~1사이의 값으로 모두 정규화하여 출력값들의 총합이 항상 1이 되도록 하는 방식이다. exp()함수로 각각의 출력값의 상대적 크기로 값을 출력시킨다.

**Visualization**

Grad-cam : Class Activation Map을 사용하면 학습한 CNN model이 입력 image의 어떤 부분을 보고 있는지를 알 수 있다. 이를 사용하기 위해서는 Convolution Layer 이후 Global Average Pooling Layer를 사용해주어야 한다는 제약조건을 가진다. 하지만 본 프로젝트에서는 Max Pooling을 사용하기 때문에 해당 제약에서 자유로운 Gradient-weighted Class Activation Map을 채택하여 시각화를 진행하였다.

t-sNE : t-SNE는 보통 고차원데이터를 2차원 또는 3차원의 저차원 데이터로 차원을 축소시켜 시각화를 진행하기 위해 사용된다. 본 프로젝트는 2차원 데이터로 차원 축소를 진행하여 데이터들을 시각화하였다. t-SNE는 t-Stochastic Neighbor Embedding의 준말로 데이터 간 거리를 stochastic probability로 변환하여 임베딩을 진행한다. 이렇게 임베딩을 진행한 후 나온 값으로 2차원으로 축소시켜준다. 축소시키는 과정에서 각 데이터들 간의 유사도를 보존하여 진행한다. 본 프로젝트는 2차원으로 변환 후 출력시켜 각각 데이터들의 유사도를 시각화시켜주었다.

**validation**

학습의 횟수이 올라갈 수록 성능이 올라가는 측면도 있지만, 동시에 overfitting이 될 가능성 또한 높아진다. 해당 경우를 방지하기 위해 callback을 통해 val\_loss(검증 손실)의 값이 10회 내로 감소하지 않을 경우 학습을 중단함으로써 overfitting을 방지하는 장치를 마련하였다.

아래는 위의 소스코드의 결과로 생성된 모델의 summary이다.

|  |
| --- |
| Model: "sequential\_2"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  conv2d\_8 (Conv2D) (None, 32, 32, 64) 1792  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_8 (MaxPooling2 (None, 16, 16, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_9 (Conv2D) (None, 16, 16, 128) 73856  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_9 (MaxPooling2 (None, 8, 8, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_10 (Conv2D) (None, 8, 8, 256) 295168  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_10 (MaxPooling (None, 4, 4, 256) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_4 (Dropout) (None, 4, 4, 256) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_11 (Conv2D) (None, 4, 4, 512) 1180160  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_11 (MaxPooling (None, 2, 2, 512) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_5 (Dropout) (None, 2, 2, 512) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  flatten\_2 (Flatten) (None, 2048) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_4 (Dense) (None, 512) 1049088  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_5 (Dense) (None, 10) 5130  =================================================================  Total params: 2,605,194  Trainable params: 2,605,194  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

첫번째 Conv2D에서는 3x3크기의 컨볼루션 필터의 개수를 64개로 설정하여 진행하였다. 그리고 padding을 same으로 설정하여 입력한 이미지 사이즈와 출력한 이미지 사이즈를 동일하게 설정해주었다. 활성화함수로는 relu를 설정하였고 커널 정규화에서는 0.001의 값으로 l2정규화를 진행하여 컨볼루션 레이어를 진행하였고 그 후 maxpooling2D를 통해 풀링을 진행하였다.

maxpooling의 결과로 ‘max\_pooling2d\_8’의 레이어를 거친 뒤 데이터의 사이즈가 (16, 16, 64)으로 감소한 것을 확인할 수 있다.

두번째 Conv2D에서는 앞서 첫번째 레이어에서 필터의 개수만 128개로 증가시켜 진행하였다.

3번째 Conv2D에서는 앞서 했던 컨볼루션 레이어에서 필터의 개수만 256개로 증가시켜 진행하였다. 그리고 과잉적합을 방지하기 위해 Dropout을 사용하였다.

4번째 Conv2D도 마찬가지로 필터의 개수만 증가시켜 진행하였고 Dropout을 사용하여 과잉적합을 방지하였다.

그 후 Flatten()를 통해 1차원 데이터로 바꿔주었고 512차원의 커널을 갖는 Fully Connected Layer를 적용해주었다.

최종적으로 클래스의 개수인 10개로 분류시키기 위해 10차원의 커널을 갖는 Fully Connected Layer을 적용시켜 분류를 진행하였다. 활성화함수로 softmax를 선택하여 진행하였다.

# **4** **Experimental Results**

## **4.1** **Test Environment**

본 프로젝트에서 사용한 라이브러리 및 작동 환경은 아래와 같다.

* tensorflw 2.5.0
* google colab

## **4.2 전처리**

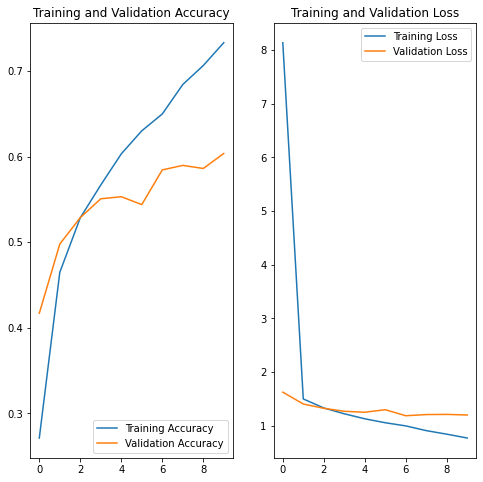
본 프로젝트에서는 데이터의 전처리를 2가지 방법으로 나누어 진행한 후 아무 처리를 하지 않은 raw 데이터와 결과를 비교하였다. 데이터 전처리에 사용한 방법은 아래와 같으며 기법의 효율성을 검증하기 위해 다른 조건은 모두 동일하게 유지하였으며(epoch=10), 각 기법들에 대한 훈련 정확도, 훈련 손실, 검증 정확도, 검증 손실을 측정하여 비교하였다.

* 최대값으로 나누기
* 표준화

### **4.2.1. raw 데이터 사용**

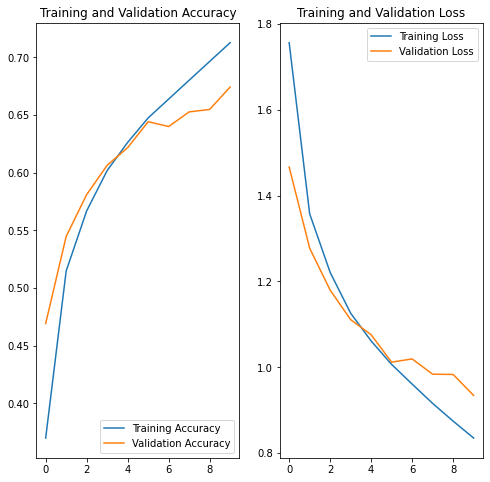
대조군으로써 주어진 데이터를 그대로 사용한 방법이다.

테스트 데이터로 측정한 해당 방법의 손실과 정확도는 각각 1.2014018297195435, 0.603600025177002이며 아래의 그래프는 epoch에 따른 정확도와 손실의 변화를 나타낸 것이다. epoch가 증가할수록 훈련 데이터의 정확도와 검증 데이터의 정확도의 차이가 커지는 것을 확인할 수 있다.



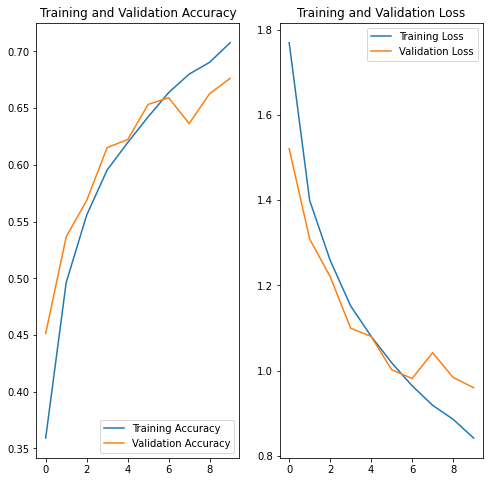
### **4.2.2 최대값으로 나누기**

각 데이터의 한 픽셀이 가질 수 있는 값인 255로 모든 값을 나눠주는 처리를 해줌으로 써 값이 스케일을 줄여주는 처리이다. 테스트 데이터로 측정한 해당 방법의 손실과 정확도는 각각 0.9523327946662903, 0.6674000024795532이다. 아래의 그래프를 통해 4.3.1보다 훈련 데이터의 정확도와 검증 데이터의 정확도의 차이가 줄어든 것을 확인할 수 있다. 테스트 데이터의 정확도가 약 0.06 상승한 것을 확인할 수 있다.



### **4.2.3 표준화**

각 데이터의 픽셀 값들에 대한 평균과 표준편차를 계산하여 모든 값들을 표준화 시킨 후 학습을 시키는 방법이다. 테스트 데이터로 측정한 해당 방법의 손실과 정확도는 각각 0.960273265838623,0.6762999892234802이다. 단순히 최대값으로 나누어 값을 스케일만 줄였을 때 보다 전체적인 정확도가 상승하였으며, 테스트 데이터의 정확도의 경우 4.2.3보다 약 0.01 정도 증가한 것을 확인할 수 있다.



전처리의 경우 값을 표준화시키는 방법이 테스트 정확도 약 0.67로 가장 좋은 성능을 보였음으로, 본 프로젝트의 모델에 표준화된 데이터를 사용하도록 하였다.

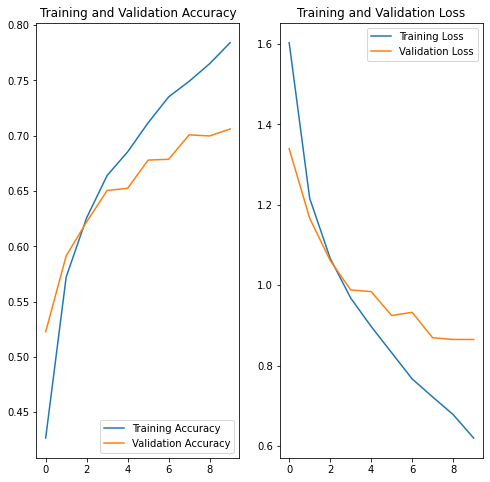
## **4.3 필터의 개수**

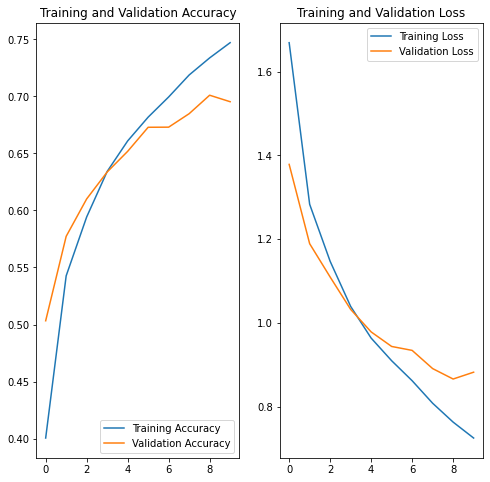
각 레이어의 필터의 개수 또한 학습에 영향을 줄 수 있다는 가정하에 레이어를 거칠수록 더 많은 필터를 적용하는 상황과 레이어를 거칠수록 적은 필터를 적용하는 상황으로 나누어 실험을 진행하였다.

총 레이어는 Fully Connected 레이어를 제외하고 3개를 레이어를 사용하였으며, 각각

* 16, 32, 64 순으로 커널의 개수를 증가
* 64, 32, 16 순으로 커널의 개수를 증가

의 상황에서 정확도와 손실을 측정하였으며 결과는 아래와 같다.





두 상황의 테스트 결과는 각각 테스트 정확도 0.7059999704360962, 0.6952999830245972로 큰 차이를 보기 어려웠다. 즉, 레이어를 거침에 따라 커널의 개수가 증가 혹은 감소하느냐는 것은 모델의 성능에 큰 영향을 미치지 않는 다는 것을 확인하였다.

### 

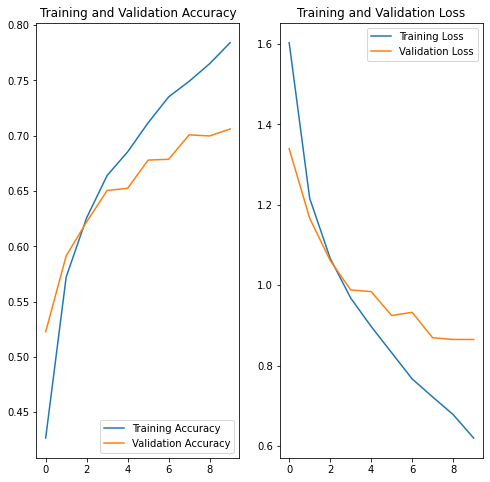
## **4.4 레이어의 개수**

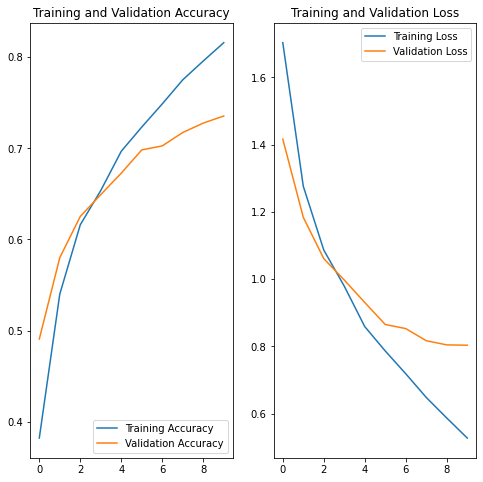
레이어를 개수에 따른 영향을 확인하기 위해 레이어의 개수에 차이를 주어 정확도를 비교하였다. 비교에는 3가지의 모델을 사용하였다. 첫번째 모델은 3개의 레이어를 갖으며 필터의 개수는 각각 16, 32, 64로 설정해주었으며, 두번째 모델은 16, 16, 32, 32, 64, 64, 세번째 모델은 순서대로 16, 16, 16, 32, 32, 32, 64, 64, 64 개의 커널을 갖는 레이어를 거치도록 설계하였다.

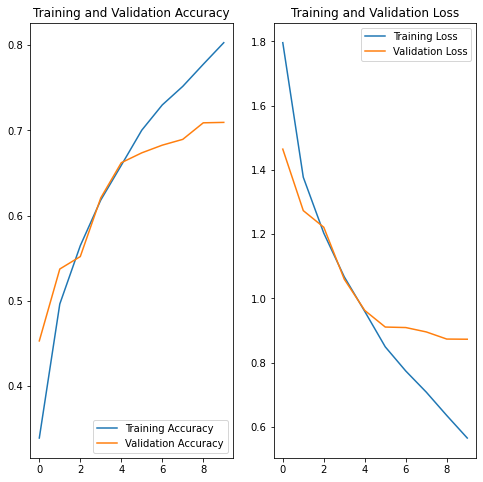
올바른 비교를 위해 epochs는 10, 커널 사이즈는 모두 3\*3, 활성화 함수는 모두 relu를 사용하도록 하였다. 세 모델 전부 레이어를 거친 뒤 512개의 커널을 갖는 fully connected 레이어와 연결하였다.

### **4.4.1 비교**

각 모델을 학습시킨 후 결과를 비교한 그래프는 아래와 같다. 순서대로 첫번째, 두번째, 세번째 모델을 가리키며, epoch가 증가함에 따른 손실과 정확도의 변화를 나타낸다.







최종적으로 측정한 모델의 정확도 및 손실은 아래와 같다. 1번 모델과 1번 모델에 레이어를 추가한 2번 모델을 비교하면, 0.03 정도의 성능향상이 있었지만 여기에 레이어를 더 추가한 3번 모델과 비교하였을 경우 오히려 성능이 떨어지는 효과를 가져왔다. 따라서 CIFAR 10 데이터를 분류하는 상황에서는 3개의 모델 중 2번 모델의 구조가 가장 적합하다는 것을 확인할 수 있었다.

|  |  |  |
| --- | --- | --- |
| 모델 | Accuracy(정확도) | Loss(손실) |
| 1 | 0.7059999704360962 | 0.8649224638938904 |
| 2 | 0.7354000210762024 | 0.8037531971931458 |
| 3 | 0.7092999815940857 | 0.8730310797691345 |

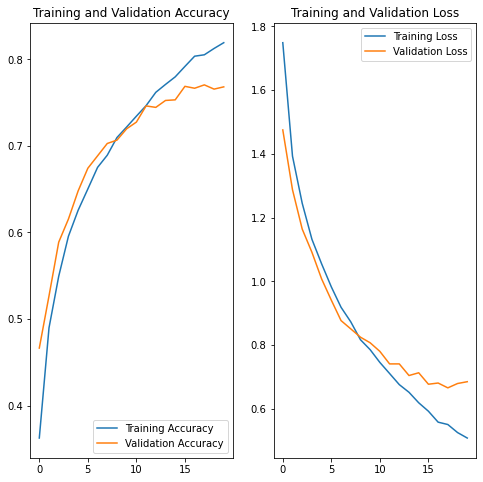
## **4.6 drop out**

본 실험에서는 레이어에 적용하는 drop out의 개수와 위치에 차이를 주어 drop out이 모델의 일반화 성능에 어떻게 영향을 미치는지 알아보았다. 케이스에 따라 1개에서 3개의 레이어에 drop out을 적용한 후 모델들의 정확도와 손실을 비교하였다.

정확한 비교를 위해 레이어의 구조 및 drop out 확률을 고정을 한 후 drop out을 적용하는 위치와 개수에만 차이를 주어 비교를 진행하였다. drop out을 실행할 확률에는 0.5를 적용하였다.

### **4.6.1 dropout을 1개의 레이어에 적용하였을 경우**

dropout을 하나의 레이어의 적용하였을 경우 epoch에 따른 정확도 및 손실의 변화는 아래의 그래프와 같다. 해당 모델의 경우 최종 손실 0.6874169707298279, 정확도 0.7608000040054321의 값을 보였다. dropout을 통해 모델의 성능이 향상된 것을 그래프로 확인할 수 있는데, epoch가 증가할 수록 훈련 정확도와 테스트 정확도의 차이가 확연했던 이전 모델들과 달리 훈련 정확도의 테스트 정확도의 차이가 약 0.05로 줄어든 것을 확인할 수 있다.



### **4.6.2 dropout을 2개의 레이어에 적용하였을 경우**

dropout을 2개의 레이어의 적용하였을 경우 최종 손실 0.656004786491394, 정확도

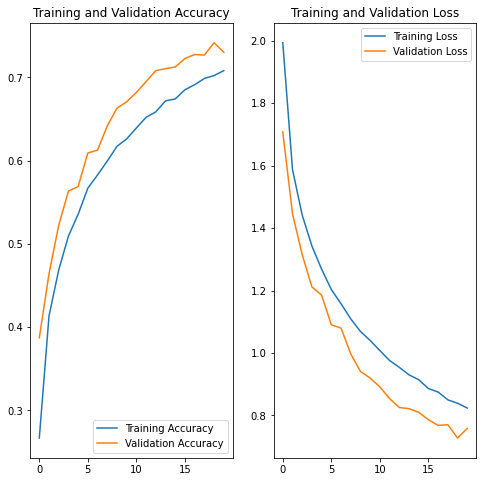
0.7705000042915344로 3개의 모델 중 가장 좋은 성능을 보였다. 해당 모델은 모든 구간에 대해서 훈련 정확도보다 검증 정확도가 높고 검증 손실은 훈련 손실보다 적은 수치를 보이고 있다.

# 

### **4.6.3 dropout을 3개의 레이어에 적용하였을 경우**

drop out을 3개의 레이어에 적용하였을 경우 테스트 데이터에 대한 손실은

0.758415162563324, 정확도는 0.7300999760627747를 보였다. 해당 모델의 경우 drop out을 1개와 2개의 레이어에 적용하였을 때 보다 더 낮은 성능을 보였는데, 이 경우 보다 많은 drop out으로 인해 underfitting 현상이 발생한 것으로 추정된다. 해당 모델의 손실과 정확도의 그래프는 아래와 같다.



최종적으로 drop out을 3개의 레이어에 적용한 모델 보다 1개의 레이어에 적용한 모델의 성능이 더 좋았고, 이 2개의 모델 보다 drop out을 2개의 레이어에 적용한 모델의 성능이 가장 좋았는데, 이는 drop out을 적용함으로써 1번 모델에 비교하여 2번 모델의 일반화 성능을 더 올릴 수 있었던 반면, 3번 모델의 경우 drop out이 과도하게 일어나 데이터를 제대로 학습 할 수 없었던 것으로 보인다.

따라서 결론적으로 drop out을 적용함으로써 모델을 성능을 향상시킬 수 있었지만, 과도하게 적용할 경우 오히려 성능을 저해하는 것을 확인할 수 있었다.

# **5** **Visualization**

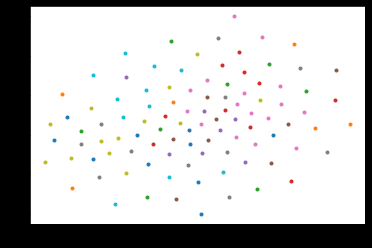
아래는 최종적으로 생성된 모델의 summary이다.

|  |
| --- |
| Model: "sequential\_2"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  conv2d\_8 (Conv2D) (None, 32, 32, 64) 1792  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_8 (MaxPooling2 (None, 16, 16, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_9 (Conv2D) (None, 16, 16, 128) 73856  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_9 (MaxPooling2 (None, 8, 8, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_10 (Conv2D) (None, 8, 8, 256) 295168  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_10 (MaxPooling (None, 4, 4, 256) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_4 (Dropout) (None, 4, 4, 256) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_11 (Conv2D) (None, 4, 4, 512) 1180160  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_11 (MaxPooling (None, 2, 2, 512) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_5 (Dropout) (None, 2, 2, 512) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  flatten\_2 (Flatten) (None, 2048) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_4 (Dense) (None, 512) 1049088  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_5 (Dense) (None, 10) 5130  =================================================================  Total params: 2,605,194  Trainable params: 2,605,194  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

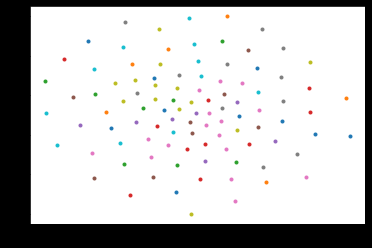
## **5.1 t-sne**

아래의 사진은 데이터가 위 모델의 convolution 레이어 거쳤을 때 각각의 feature map을 거쳤을 때를 t-sne을 통해 2차원으로 시각화한 이미지이다. 시각화에는 테스트 데이터 중 0번부터 99번의 데이터를 표본 추출하여 나타내었다.

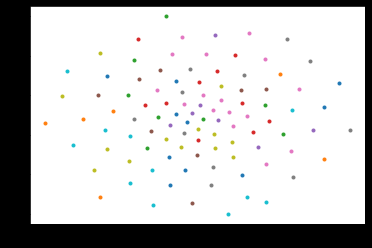
사진은 순서 대로 데이터가 conv2d\_8, conv2d\_9, conv2d\_10, conv2d\_11의 레이어를 거친 후 feature space내에 어떻게 분포하고 있는지는 나타내는 것이며, 같은 클래스에 속하는 데이터끼리는 같은 색을 가지고 있다. 사진을 순서대로 확인을 해 보면 가장 첫 이미지의 경우 여러 색깔이 여러군데에 분포되어 있는 것을 확인할 수 있지만, conv2d\_11를 거친 후의 이미지를 확인하면같은 색깔의 점들이 이전의 이미지들에 비해 확연하게 몰려있는 것을 확인할 수 있다.



<conv2d\_8를 거친 후의 feature space를 시작화한 이미지>



<conv2d\_9를 거친 후의 feature space를 시작화한 이미지>



<conv2d\_10를 거친 후의 feature space를 시작화한 이미지>



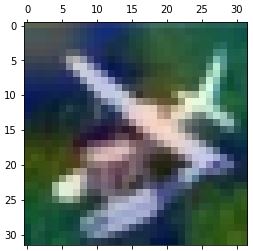
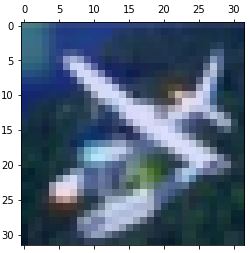
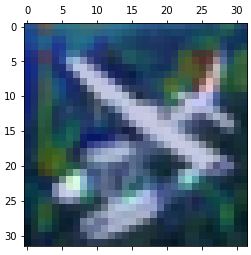
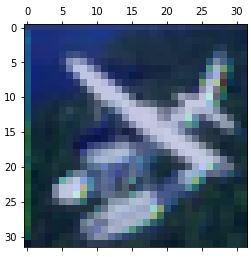
<conv2d\_11를 거친 후의 feature space를 시작화한 이미지>

## 

## **5.2 grad-CAM**

먼저, colormap으로는 ‘jet’를 사용하였다.

위 사진에서처럼 우측색에 가까워질수록 학습된 모델이 좀 더 중요한 특징으로 생각하는 부분이다.



위 사진은 모델에 존재하는 4개의 convolution layer를 순서대로 촬영한 것이다. 좌측 상단의 사진이 ‘conv2d\_8’, 우측 상단의 사진이 ‘conv2d\_9’, 좌측 하단의 사진이 ‘conv2d\_10’, 우측 하단의 사진이 ‘conv2d\_11’에 해당한다. layer가 출력층에 가까이 다가갈수록 입력 이미지에서 보다 확실하게 특징을 잡아가는 것을 알 수 있다.

# **6** **Discussion and Conclusion**

본 프로젝트에서는 전처리 방법, 필터의 개수, 레이어의 개수, 학습 횟수(epoch), dropout 의 유무 및 개수를 비교하여 각각 모델의 성능에 어떤 영향을 미치는 지를 확인하였다. 그리고 학습된 모델에서 레이어를 거침에 따라 데이터가 어떻게 변화하는지 와 특징이 검출되는지를 t-SNE와 Grad-CAM을 사용하여 시각화를 진행하였다.

전처리 과정에서는 데이터를 그대로 사용하는 방법과, 먼저 전처리 과정은 픽셀 값들에 대한 평균과 표준편차를 계산하여 각 픽셀값에서 평균을 빼 표준편차값을 나누어 주는 표준화 방식을 비교하였으며, 그 중 데이터를 표준화하는 방법이 가장 좋은 성능을 보였다.

그리고 레이어의 수는 conv2D레이어 4개, maxPooling2D레이어 4개, Dropout 레이어 2개와 Flatten레이어, dense레이어 2개로 프로젝트를 진행하였다.

필터의 수에서는 각 Conv2D마다 64, 128, 256, 512로 설정하여 진행하였다.

epoch는 100으로 설정한 후, 학습이 진행되면서 validation loss가 10회 이상 줄어들 지 않을 경우, 학습을 중단시키는 방식이 가장 좋은 값을 보여주었기 때문에 epoch는 위 방법으로 진행하였다.

dropout에서는 앞서 4.6에서 나와있듯이 dropout을 2번 사용할 경우 가장 좋은 accuracy를 보여주어 2번으로 설정하였다.

위 결과를 취합하여 3에서 제시한 구조로 학습을 시켜 모델을 생성하였으며 최종적으로 0.8317000269889832의 정확도의 성능의 모델을 얻을 수 있었다.