CNN 분류기
개발 프로젝트

전자정보공학부 IT융합전공 박준혁 (20170590)
AI융합학부 김혜빈 (20192897)
AI융합학부 이찬도 (20160391)

목차

Team Introduction

Role Sharing

[역할 분담]

- 팀장 박준혁 : 데이터 불러오기 코드 작성 / 데이터셋 통합 코드 작성 / 데이터 전처리 코드 작성 / CNN 구조 설계 / 평가보고서 작성 / 발표 자료 작성 / 발표 및 시연
- 팀원 김혜빈 : 데이터 전처리 코드 작성 (데이터 증강, Shuffle) / CNN 구조 설계 / CNN 구조 파라미터 조정 / 모델 학습 및 검정 / 과대적합 문제 해결
- 팀원 이찬도 : NVIDIA AGX Xavier Developer Kit 활용 방법 조사 및 예제 실행 / CheckPointAndSaveModel 방법 조사 / Helper 코드 활용 방법 조사

Presenter

Programmer

Engineer

Meeting Progress

[회의 진행]

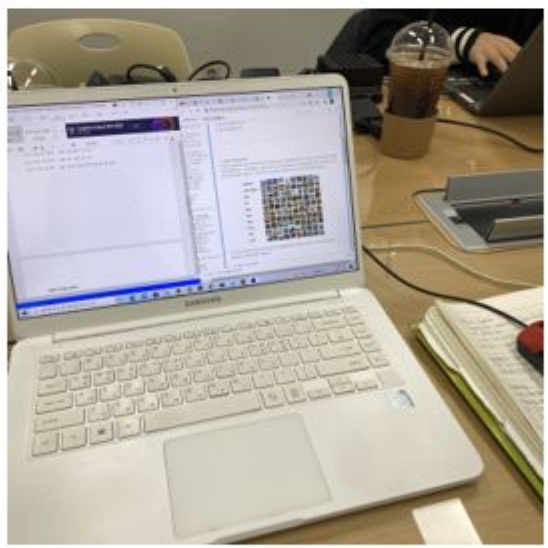
“2021.11.22” 1차 회의 : 전체 프로젝트 계획 수립 및 To Do List 작성

“2021.11.25” 2차 회의 : 데이터 불러오기, 데이터셋 통합 코드 작성 및 점검

“2021.11.28” 3차 회의 : CNN 구조 설계 1차 시도

“2021.12.01” 4차 회의 : CNN 구조 설계 2차 시도

“2021.12.05” 5차 회의 : 평가 보고서, 발표 자료 작성 및 시연 준비

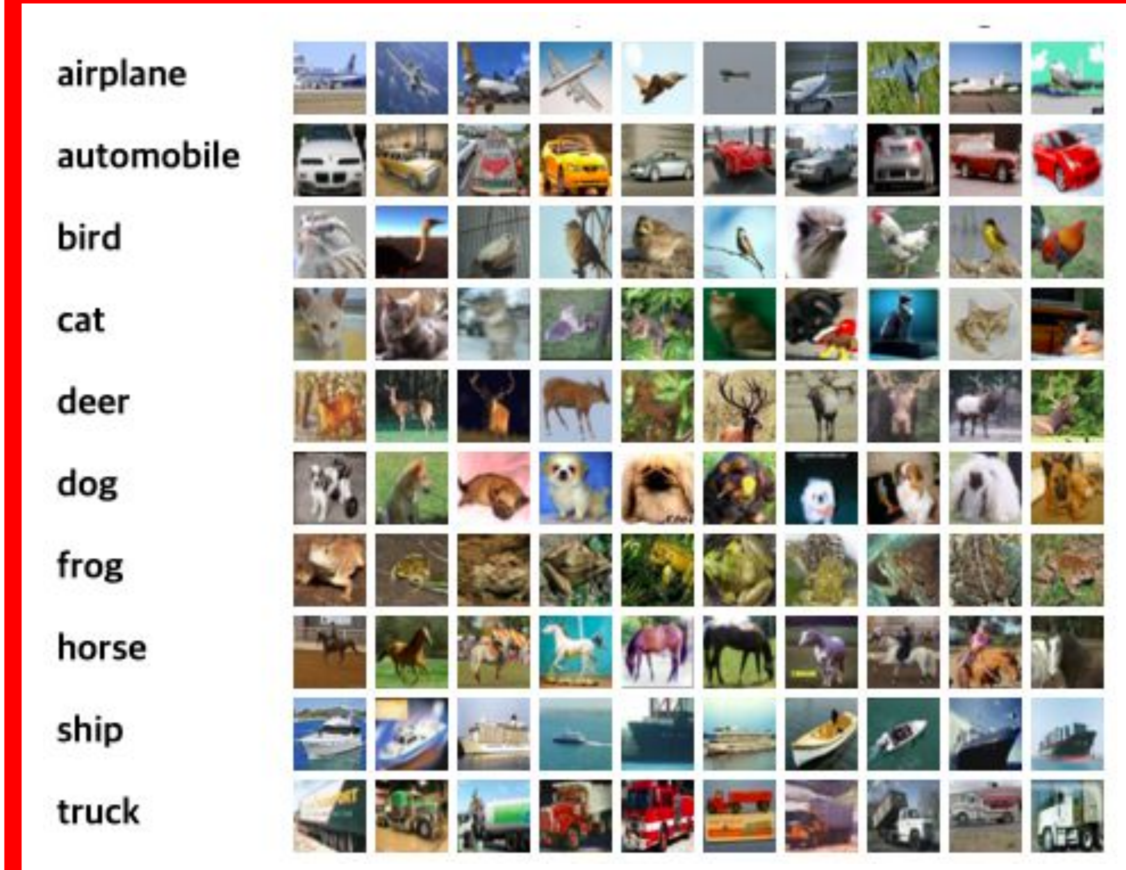


이름

- 211122_1차회의.txt
- 211125_2차회의.txt
- 211128_3차회의.txt
- CNN 개발 프로젝트 발표자료.pptx
- 결과보고서.docx
- 발표자료.pptx
- 프로젝트 공지.txt

Data Preparation

CIFAR–10 Dataset



- 데이터 수집 / 준비

CIFAR-10 dataset은 32x32 픽셀의 60,000개 컬러 이미지가 10개의 클래스로 레이블링 되어 있음.

각 카테고리별 6,000개의 데이터 중 앞에 5,000개는 훈련 데이터, 뒤에 1,000개는 시험 데이터로 사용함.

이렇게 해서 총 60,000개의 이미지를 통해 CNN 분류기를 설계함.

Data Preparation

CIFAR–10 Dataset

아래 그림은 CIFAR 10 데이터 중 무작위로 40개의 이미지를 추출하여 출력한 결과와 함께 한 이미지의 픽셀 값을 나타낸 것이다. CIFAR 10 데이터셋 내 모든 인스턴스들은 (32, 32, 3)의 구조를 가지고 있는데, 이는 가로 세로 픽셀 값이 모두 '32'이며, 3개의 RGB 채널을 가지고 있는 것을 의미한다. 또한, 각 픽셀 값은 Resize로 이미지 사이즈가 조정되는 과정에서 0 ~ 1 사이의 값으로 변환되기 때문에 입력 단계에서 추가적인 정규화 처리는 필요하지 않다.

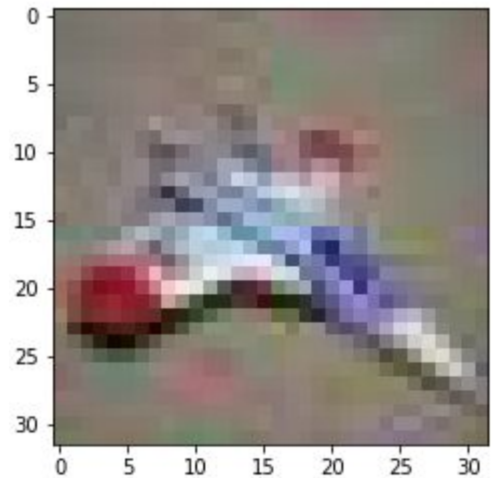
```
array([[0.78431373, 0.79215686, 0.77254902],
       [0.79215686, 0.8       , 0.78039216],
       [0.79607843, 0.80392157, 0.78431373],
       [0.79607843, 0.80392157, 0.78431373],
       [0.81176471, 0.81960784, 0.8       ],
       [0.83137255, 0.83921569, 0.81960784],
       [0.83529412, 0.84313725, 0.82352941],
       [0.82352941, 0.83137255, 0.81176471],
       [0.79215686, 0.8       , 0.78039216],
       [0.86666667, 0.8745098 , 0.85490196],
       [0.87058824, 0.87843137, 0.85882353],
       [0.80784314, 0.81568627, 0.79607843],
       [0.85098039, 0.85882353, 0.83921569],
       [0.88627451, 0.89411765, 0.8745098 ],
```

Data Load

Definition

- **기본 변수 정의**

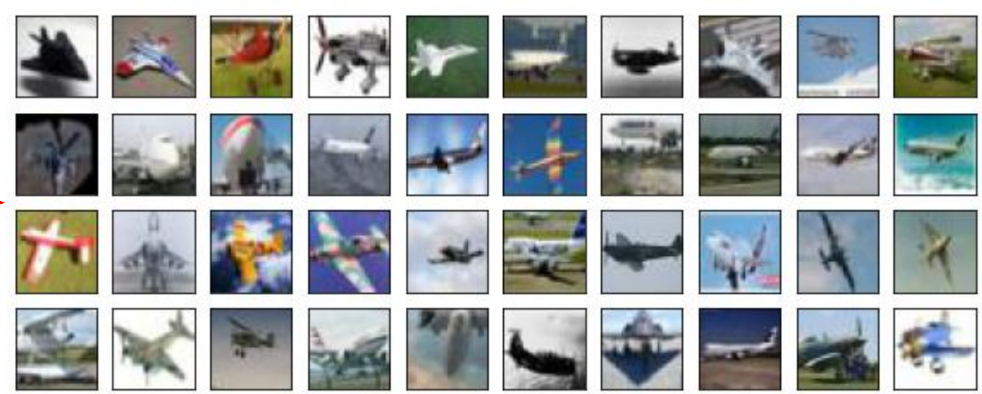
```python
# 상수 사용을 위함
class Default:
    imgR      = 32
    imgC      = 32
    channel   = 3
    imgCount  = 6000
    split     = 5000
```




```
array([[0.78431373, 0.79215686, 0.77254902],
       [0.79215686, 0.8       , 0.78039216],
       [0.79607843, 0.80392157, 0.78431373],
       [0.79607843, 0.80392157, 0.78431373],
       [0.81176471, 0.81960784, 0.8       ],
       [0.83137255, 0.83921569, 0.81960784],
       [0.83529412, 0.84313725, 0.82352941],
       [0.82352941, 0.83137255, 0.81176471],
       [0.79215686, 0.8       , 0.78039216],
       [0.86666667, 0.8745098 , 0.85490196],
       [0.87058824, 0.87843137, 0.85882353],
       [0.80784314, 0.81568627, 0.79607843],
```

- **사용자 함수 정의**

```python
# Visualize Image
def plot_images(nRow,nCol,img):
    fig = plt.figure()
    fig,ax = plt.subplots(nRow,nCol,figsize=(nCol,nRow))
    for i in range(nRow):
        for j in range(nCol):
            if nRow <= 1: axis = ax[j]
            else:         axis = ax[i,j]
            axis.get_xaxis().set_visible(False)
            axis.get_yaxis().set_visible(False)
            axis.imshow(img[i*nCol+j])
```



Data Load

Dataset Create & Train Test Split

```
# 데이터셋 로드
(train_images, train_labels), (test_images, test_labels) = data_load(path, class_labels)

print(f'train data: {train_images.shape} {train_labels.shape}')
print(f'test data: {test_images.shape} {test_labels.shape}')
```

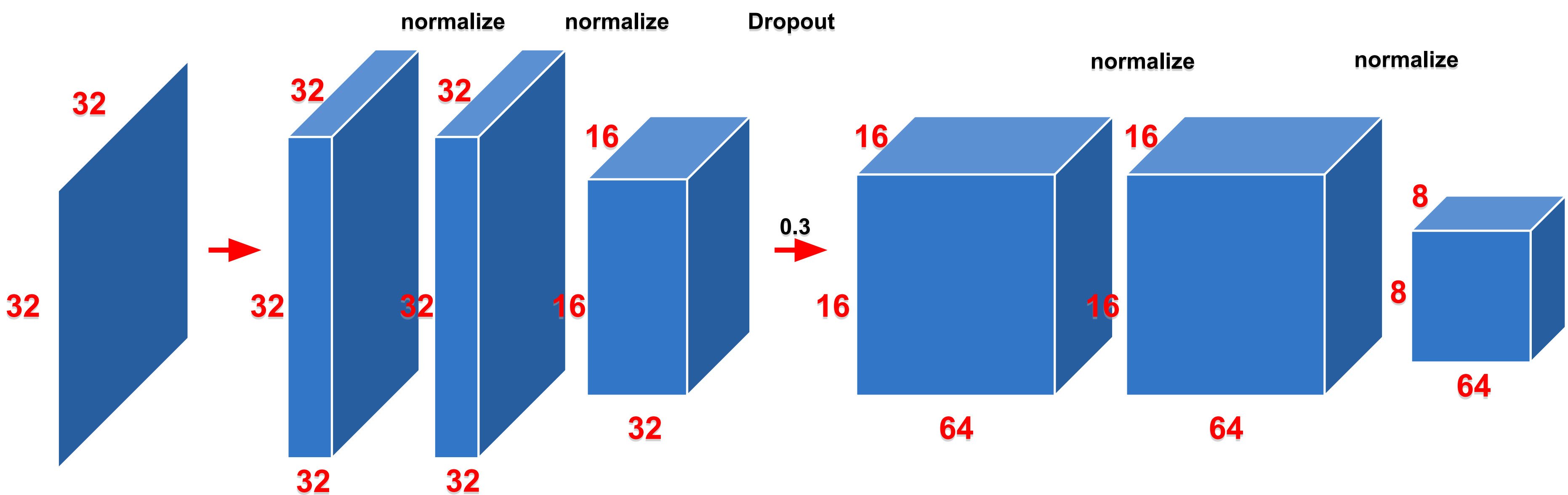
```
0: (6000, 32, 32, 3) (6000,)
1: (6000, 32, 32, 3) (6000,)
2: (6000, 32, 32, 3) (6000,)
3: (6000, 32, 32, 3) (6000,)
4: (6000, 32, 32, 3) (6000,)
5: (6000, 32, 32, 3) (6000,)
6: (6000, 32, 32, 3) (6000,)
7: (6000, 32, 32, 3) (6000,)
8: (6000, 32, 32, 3) (6000,)
9: (6000, 32, 32, 3) (6000,)
Done: set x, y dictionary
```

```
train 0: (5000, 32, 32, 3) (5000,)
test 0: (1000, 32, 32, 3) (1000,)
train 1: (5000, 32, 32, 3) (5000,)
test 1: (1000, 32, 32, 3) (1000,)
train 2: (5000, 32, 32, 3) (5000,)
test 2: (1000, 32, 32, 3) (1000,)
train 3: (5000, 32, 32, 3) (5000,)
test 3: (1000, 32, 32, 3) (1000,)
train 4: (5000, 32, 32, 3) (5000,)
test 4: (1000, 32, 32, 3) (1000,)
train 5: (5000, 32, 32, 3) (5000,)
test 5: (1000, 32, 32, 3) (1000,)
train 6: (5000, 32, 32, 3) (5000,)
test 6: (1000, 32, 32, 3) (1000,)
train 7: (5000, 32, 32, 3) (5000,)
test 7: (1000, 32, 32, 3) (1000,)
train 8: (5000, 32, 32, 3) (5000,)
test 8: (1000, 32, 32, 3) (1000,)
train 9: (5000, 32, 32, 3) (5000,)
test 9: (1000, 32, 32, 3) (1000,)
Done: split x,y dict --> train/test
```

```
Done: set x, y data
Successfully load data
train data: (50000, 32, 32, 3) (50000,)
test data: (10000, 32, 32, 3) (10000,)
```

CNN Model Design & Learning

CNN Structure Block Diagram (1)



Input Layer

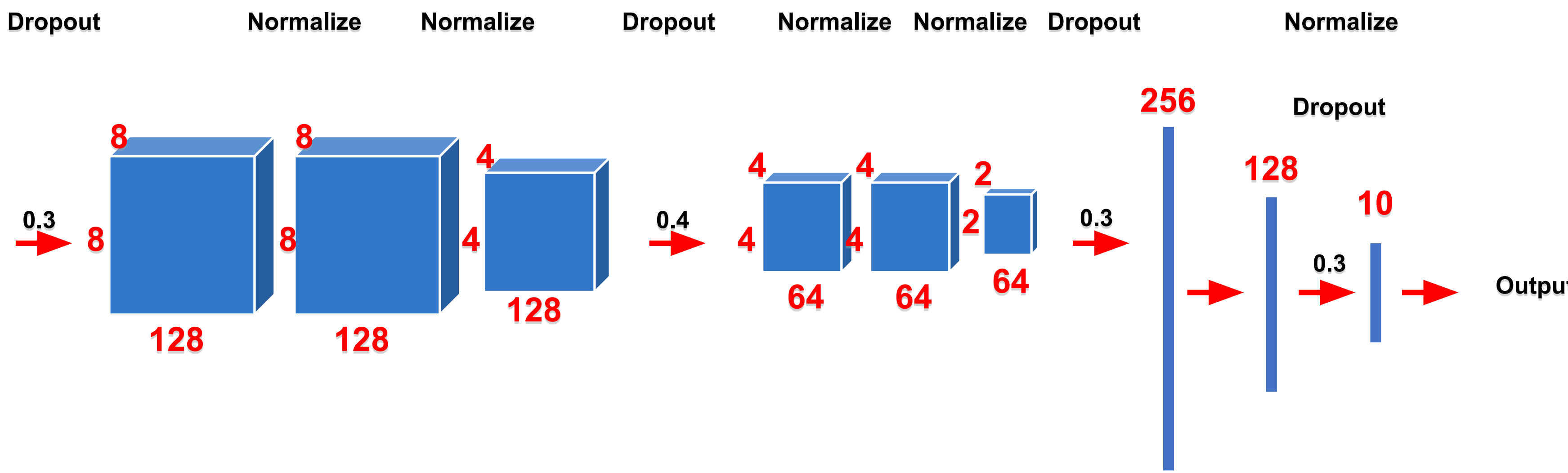
Convolution
Pooling Layer1
(Conv1 / Conv2 / Pool1)

Convolution
Pooling Layer2
(Conv3 / Conv4 / Pool2)

3	32	32	32	64	64	64
*	*	*	*	*	*	*
32	32	32	16	16	16	8
*	*	*	*	*	*	*
32	32	32	16	16	16	8

CNN Model Design & Learning

CNN Structure Block Diagram (2)



Convolution Pooling Layer3

(Conv5 / Conv6 / Pool3)

Convolution Pooling Layer4

(Conv7 / Conv8 / Pool4)

Dense Output Layer

(Flatten / Dense1 / Dense2)

128	128	128	64	64	64	256	128	10
*	*	*	*	*	*	*	*	*
8	8	4	4	4	2	1	1	1
*	*	*	*	*	*	*	*	*
8	8	4	4	4	2	1	1	1

CNN Model Design & Learning

CNN Structure Summary

Model: "sequential_18"

Layer (type)	Output Shape	Param #
conv2d_146 (Conv2D)	(None, 32, 32, 32)	896
batch_normalization_164 (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_147 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_165 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_73 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_108 (Dropout)	(None, 16, 16, 32)	0
conv2d_148 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_166 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_149 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_167 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_74 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_109 (Dropout)	(None, 8, 8, 64)	0
conv2d_150 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_168 (BatchNormalization)	(None, 8, 8, 128)	512
dropout_110 (Dropout)	(None, 8, 8, 128)	0
conv2d_151 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_169 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_75 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_111 (Dropout)	(None, 4, 4, 128)	0
conv2d_152 (Conv2D)	(None, 4, 4, 64)	73792
batch_normalization_170 (BatchNormalization)	(None, 4, 4, 64)	256
dropout_112 (Dropout)	(None, 4, 4, 64)	0
conv2d_153 (Conv2D)	(None, 4, 4, 64)	36928
batch_normalization_171 (BatchNormalization)	(None, 4, 4, 64)	256
max_pooling2d_76 (MaxPooling2D)	(None, 2, 2, 64)	0
dropout_113 (Dropout)	(None, 2, 2, 64)	0
flatten_18 (Flatten)	(None, 256)	0
dense_36 (Dense)	(None, 128)	32896
batch_normalization_172 (BatchNormalization)	(None, 128)	512
dropout_114 (Dropout)	(None, 128)	0
dense_37 (Dense)	(None, 10)	1290

Total params: 434,730
Trainable params: 433,322
Non-trainable params: 1,408

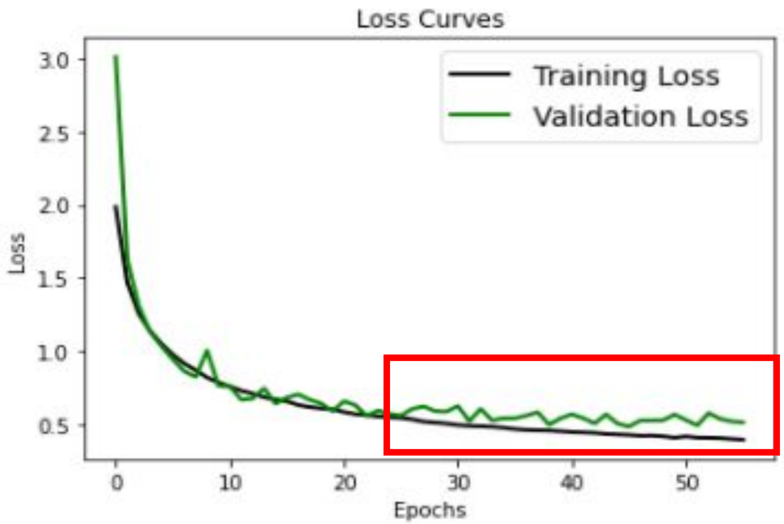
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

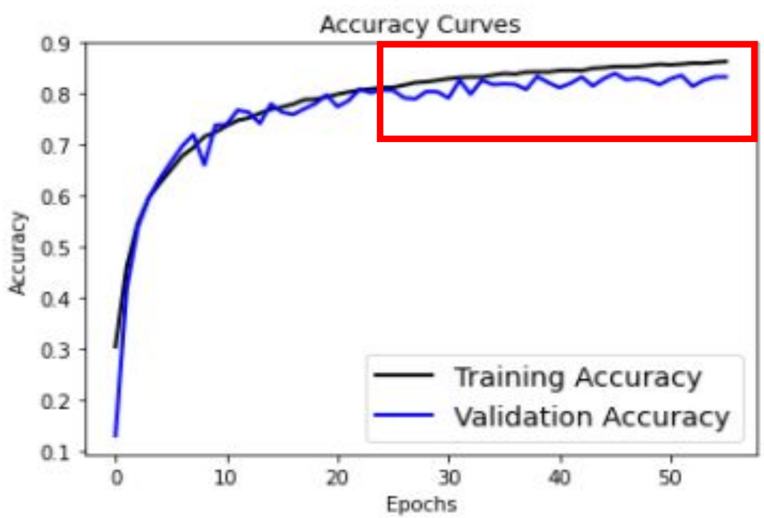
CNN Model Design & Learning

CNN Model Learning

- **Learning Parameter : Epochs = 500 / Early Stopping.Patience = 10**

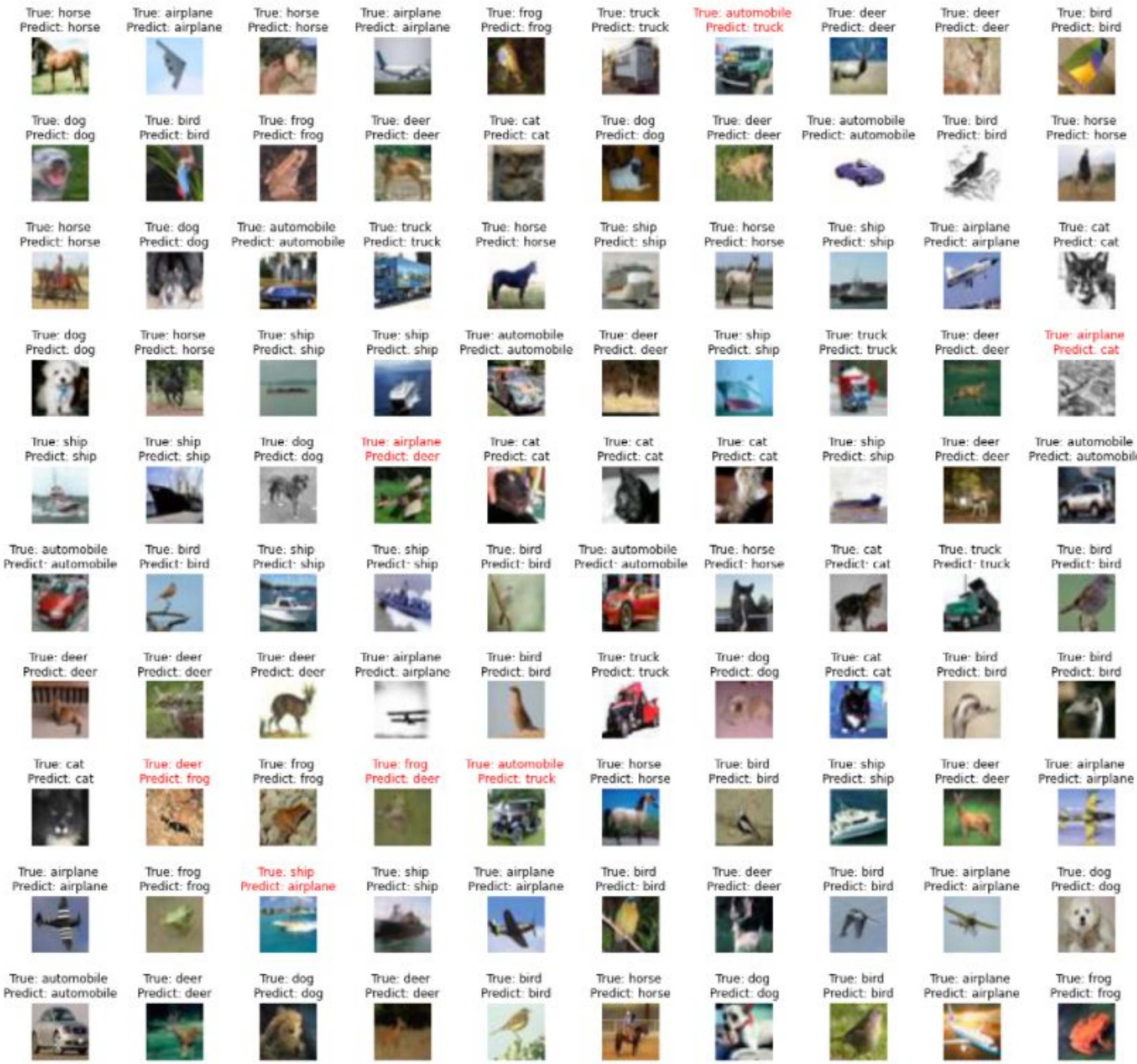
```
Epoch 00044: val_loss did not improve from 0.49674

Epoch 00045: val_loss did not improve from 0.49674

Epoch 00046: val_loss improved from 0.49674 to 0.48407,
000_training/cp-0046-0.48.ckpt

Epoch 00047: val_loss did not improve from 0.48407

Epoch 00048: val_loss did not improve from 0.48407

Epoch 00049: val_loss did not improve from 0.48407

Epoch 00050: val_loss did not improve from 0.48407

Epoch 00051: val_loss did not improve from 0.48407

Epoch 00052: val_loss did not improve from 0.48407

Epoch 00053: val_loss did not improve from 0.48407
--------------------------------------------------
Epoch 00054: val_loss did not improve from 0.48407

Epoch 00055: val_loss did not improve from 0.48407

Epoch 00056: val_loss did not improve from 0.48407
```





Evaluation

Performance Test



- **Prediction**

```
pred = model.predict(test_images)
print(pred)
print(pred.shape)

# Converting the predictions into label index
pred_labels = np.argmax(pred, axis=1)
print(pred_labels)
```

```
[[8.1475837e-06 1.6372454e-07 1.2896578e-04 ... 1.7792576e-04
  1.1053589e-05 3.8428734e-06]
 [1.6674405e-03 3.7332320e-05 4.9921684e-03 ... 1.0810441e-04
  8.9227957e-05 2.7710406e-04]
 [5.5045797e-04 1.1152675e-05 3.3990669e-01 ... 5.2811165e-04
  4.9362170e-05 1.2164089e-04]
 ...
 [3.6471534e-05 1.6703419e-06 6.3088536e-04 ... 7.7230132e-01
  2.8163120e-06 2.7078061e-06]
 [3.7712550e-03 5.0671010e-06 9.8829460e-01 ... 3.9528991e-04
  5.2363132e-05 2.0366554e-04]
 [6.8038549e-05 1.2875648e-05 9.7700185e-01 ... 6.9528673e-05
  4.6684770e-05 1.1698684e-05]]
(10000, 10)
[3 3 4 ... 7 2 2]
```

- **Result**

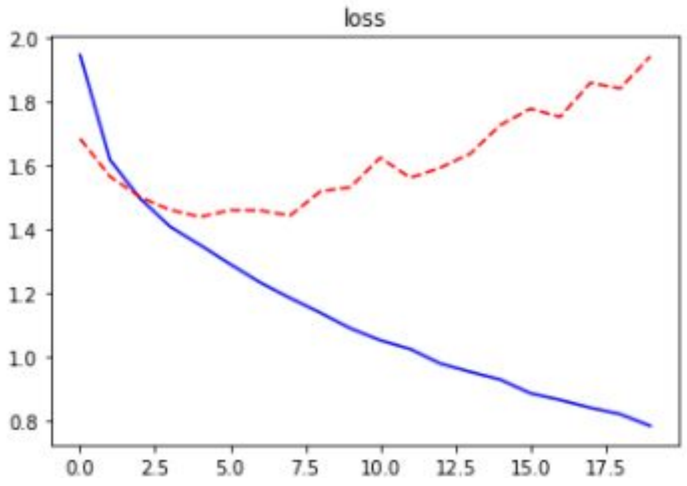
```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('테스트 정확도:',test_acc)
```

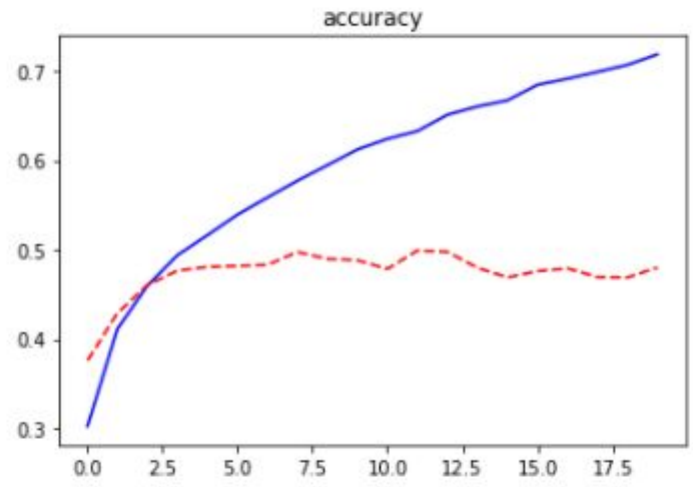
```
313/313 - 20s - loss: 0.2993 - accuracy: 0.9041 - 20s/epoch - 63ms/step
테스트 정확도: 0.9041000008583069
```

테스트 정확도 : 0.904

Conclusion

- **CNN 구조 설계의 중요성**





하이퍼 파라미터 조정 없이 Convolution Pooling Layer를 두 개 층만 쌓아 만들었던 초기 모델은 좋은 분류 성능을 보이지 못하였음. 정확도가 0.4 ~ 0.7 사이에서 형성되며, loss 값 역시 튀는 현상이 목격되었음.

따라서 보다 복잡한 CNN 구조를 설계하여 조금씩 파라미터를 조정해가는 방식의 접근이 필요함을 깨달음.

Convolution Pooling Layer를 4개 층까지 확장하고, BatchNormalization, DropOut 기법을 사용하여, 오차 값을 보정하고 과대적합 문제를 해결함.

그 결과, 테스트 데이터 정확도 : 0.904의 높은 분류 성능을 얻음.

Thank you