

# Report



## 2021 딥러닝/클라우드 기말고사 대체과제

작성자	소프트웨어학과 김채은
작성일	2021. 12. 18.
담당교수	오세종 교수님

## ■ 개요

### 1. 데이터 셋 준비

- 이미지를 클래스 별로 분류한다.
- 이미지를 전처리한다.
- 훈련 셋과 테스트 셋을 나눈다.

### 2. 훈련 모델 만들기

- 레이어를 추가하며 훈련 모델을 만든다.
- 모델을 학습시키고 성능을 평가한다.

### 3. 이미지 사이즈 별 accuracy 비교

- 이미지 사이즈에 따른 accuracy 차이가 있는지 비교해본다.

### 4. 소감

## 1. 데이터 셋 준비

모델을 만들기 전 이미지 전처리를 해야 한다. 먼저 클래스 별로 데이터를 분류한다.

```
import os
from sklearn.model_selection import train_test_split
import shutil

# 데이터 셋 경로
path = '/content/drive/MyDrive/Colab Notebooks/dataset2'

# 분류 클래스
label = ['sunrise', 'shine', 'rain', 'cloudy']

for i in label:
    # 데이터 셋에 있는 jpg 파일 리스트로 받기
    train = os.listdir(path)
    for j in train:
        # jpg 가 아닌 것(폴더) 패스
        if 'jpg' not in j:
            continue
        # label 이 파일 이름에 있으면
        if i in j:
            # dataset2/파일.jpg 에서 dataset2/train/label/파일.jpg 로 폴더 이동
            src = path+'/' +j
            dir = path+'/train/'+i+'/' +j
            shutil.move(src, dir)
```

코드를 실행한 후 디렉터리 구조이다. 클래스 별로 데이터가 분류되어 들어있다.

```
dataset2
├── train
│   ├── cloudy
│   ├── rain
│   ├── shine
│   └── sunrise
```

이미지 사이즈를 조정하고, RGB로 이미지 객체를 변환하는 전처리 작업을 한다. 그 다음 train, test set 나눠서 저장한다. 비율은 7:3으로 한다.

```
import glob
from PIL import Image
import numpy as np
import os
from sklearn.model_selection import train_test_split

path = '/content/drive/MyDrive/Colab Notebooks/dataset2'

label = ['sunrise', 'shine', 'rain', 'cloudy']
class_len = len(label)

train_path = path+'/train'

# 이미지 크기 설정
img_width = 300
img_height = 300

pixels = img_width*3 + img_height

x = []
y = []

for i, lab, in enumerate(label):
    # label을 배열로 구분하게 한다.
    # [1, 0, 0, 0] / [0, 1, 0, 0] / [0, 0, 1, 0] / [0, 0, 0, 1]
    label_arr = [0 for _ in range(class_len)]
    label_arr[i] = 1

    dir = train_path + '/' + lab

    # label 디렉터리에 있는 jpg 파일 모두 불러오기
    files = glob.glob(dir + '/*.jpg')

    for j, file in enumerate(files):
        img = Image.open(file)
        # 이미지 객체를 RGB로 변환
        img = img.convert('RGB')

        # 위에서 정한 넓이, 높이로 이미지 사이즈 변환
        img = img.resize((img_width, img_height))

        # 이미지를 numpy 배열로
        data = np.asarray(img)
```

```

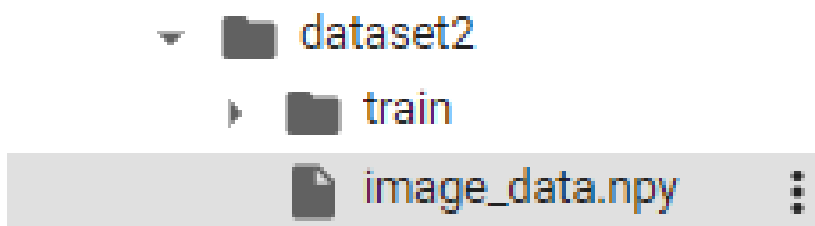
# x 에는 이미지, y 에는 label
x.append(data)
y.append(label_arr)

X = np.array(x)
Y = np.array(y)

# train set, test set 나누기
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=123)

# npy 로 저장 (저장 안해도 o)
sets = (X_train, X_test, Y_train, Y_test)
np.save(path+"/image_data.npy", sets)

```



npz 파일이 저장되었다. 이것을 불러와서 train, test 셋으로 사용한다.

```

# 저장한 numpy 배열 불러오기
X_train, X_test, y_train, y_test = np.load(path + '/image_data.npy',
allow_pickle=True)

```

npz 파일을 불러올 때 allow\_pickle=True 안하면 다음과 같은 오류가 난다.

```

-----
ValueError                                Traceback (most recent call last)
<ipython-input-15-ee7d62da6cd> in <module>()
----> 1 X_train, X_test, y_train, y_test = np.load(path + '/image_data.npy')
      2
      3 print(X_train.shape)
      4 print(X_train.shape[0])
      5 print(np.bincount(y_train))

1 frames
/usr/local/lib/python3.7/dist-packages/numpy/lib/format.py in read_array(fp, allow_pickle, pickle_kwargs)
    725     # The array contained Python objects. We need to unpickle the data.
    726     if not allow_pickle:
--> 727         raise ValueError("Object arrays cannot be loaded when "
    728                         "allow_pickle=False")
    729     if pickle_kwargs is None:

```

ValueError: Object arrays cannot be loaded when allow\_pickle=False

SEARCH STACK OVERFLOW

## 2. 훈련모델 만들기

### (1) 첫 번째 모델

모델을 만든다. 초기모델은 다음과 같다. 에포크를 10으로 두고 모델을 구성한 다음 accuracy 추이를 보고 에포크를 늘려 학습을 할 생각이다.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt

X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

model = Sequential()
model.add(Conv2D(32, (3,3), padding="same", input_shape=X_train.shape[1:], activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(4, activation="sigmoid"))

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

model_path = path + "/weather.model"

checkpoint = ModelCheckpoint(filepath=model_path, monitor='val_loss', verbose=1, save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', patience=7)
```

## 첫 번째 모델 구조

```
model.summary()
```

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_9 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_9 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_10 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_10 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_6 (Dropout)	(None, 8, 8, 64)	0
conv2d_11 (Conv2D)	(None, 8, 8, 64)	36928
max_pooling2d_11 (MaxPooling2D)	(None, 4, 4, 64)	0
dropout_7 (Dropout)	(None, 4, 4, 64)	0
flatten_2 (Flatten)	(None, 1024)	0
dense_4 (Dense)	(None, 256)	262400
dropout_8 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 4)	1028

=====  
Total params: 328,996  
Trainable params: 328,996  
Non-trainable params: 0  
=====

## 첫 번째 모델 accuracy

```
Epoch 6/10
21/21 [=====] - ETA: 0s - loss: 0.5784 - accuracy: 0.3338
Epoch 00006: val_loss improved from 0.68527 to 0.68344, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 419ms/step - loss: 0.5784 - accuracy: 0.3338 - val_loss: 0.6834 - val_accuracy: 0.2941
Epoch 7/10
21/21 [=====] - ETA: 0s - loss: 0.5711 - accuracy: 0.3293
Epoch 00007: val_loss improved from 0.68344 to 0.68170, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 451ms/step - loss: 0.5711 - accuracy: 0.3293 - val_loss: 0.6817 - val_accuracy: 0.2941
Epoch 8/10
21/21 [=====] - ETA: 0s - loss: 0.5565 - accuracy: 0.3907
Epoch 00008: val_loss improved from 0.68170 to 0.67990, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 421ms/step - loss: 0.5565 - accuracy: 0.3907 - val_loss: 0.6799 - val_accuracy: 0.2605
Epoch 9/10
21/21 [=====] - ETA: 0s - loss: 0.5475 - accuracy: 0.4222
Epoch 00009: val_loss improved from 0.67990 to 0.67798, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 465ms/step - loss: 0.5475 - accuracy: 0.4222 - val_loss: 0.6780 - val_accuracy: 0.2605
Epoch 10/10
21/21 [=====] - ETA: 0s - loss: 0.5568 - accuracy: 0.4087
Epoch 00010: val_loss improved from 0.67798 to 0.67577, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 8s 401ms/step - loss: 0.5568 - accuracy: 0.4087 - val_loss: 0.6758 - val_accuracy: 0.2605
```

## (2) 두 번째 모델

레이어 하나를 추가했다. 에포크는 앞과 같이 10으로 두고 학습을 실행했다.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt

X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

model = Sequential()
model.add(Conv2D(32, (3,3), padding="same", input_shape=X_train.shape[1:], activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(4, activation="sigmoid"))

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

model_path = path + "/weather.model"

checkpoint = ModelCheckpoint(filepath=model_path, monitor='val_loss',
    verbose=1, save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', patience=7)
```



## 두 번째 모델 구조



```
model.summary()
```



```
Model: "sequential_4"
```

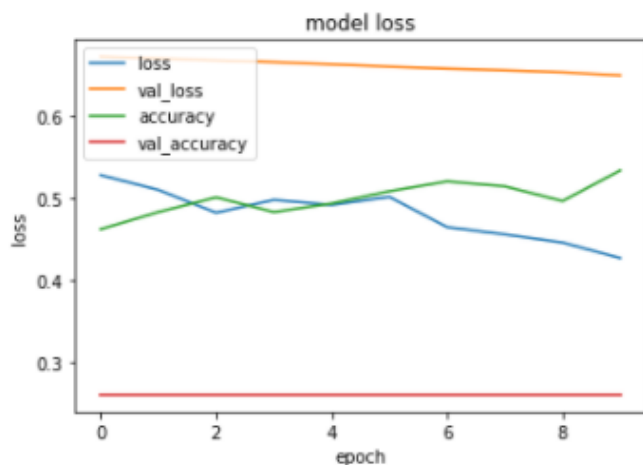
Layer (type)	Output Shape	Param #
conv2d_17 (Conv2D)	(None, 300, 300, 32)	896
max_pooling2d_16 (MaxPooling2D)	(None, 150, 150, 32)	0
conv2d_18 (Conv2D)	(None, 150, 150, 32)	9248
max_pooling2d_17 (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_19 (Conv2D)	(None, 75, 75, 64)	18496
max_pooling2d_18 (MaxPooling2D)	(None, 37, 37, 64)	0
dropout_11 (Dropout)	(None, 37, 37, 64)	0
conv2d_20 (Conv2D)	(None, 37, 37, 128)	73856
max_pooling2d_19 (MaxPooling2D)	(None, 18, 18, 128)	0
dropout_12 (Dropout)	(None, 18, 18, 128)	0
conv2d_21 (Conv2D)	(None, 18, 18, 256)	295168
max_pooling2d_20 (MaxPooling2D)	(None, 9, 9, 256)	0
dropout_13 (Dropout)	(None, 9, 9, 256)	0
flatten_3 (Flatten)	(None, 20736)	0
dense_6 (Dense)	(None, 512)	10617344
dropout_14 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 4)	2052
Total params: 11,017,060		
Trainable params: 11,017,060		
Non-trainable params: 0		

## 두 번째 모델 accarcy

```
Epoch 6/10
21/21 [=====] - ETA: 0s - loss: 0.5021 - accuracy: 0.5090
Epoch 00006: val_loss improved from 0.66403 to 0.66115, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 444ms/step - loss: 0.5021 - accuracy: 0.5090 - val_loss: 0.6611 - val_accuracy: 0.2605
Epoch 7/10
21/21 [=====] - ETA: 0s - loss: 0.4651 - accuracy: 0.5210
Epoch 00007: val_loss improved from 0.66115 to 0.65866, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 10s 487ms/step - loss: 0.4651 - accuracy: 0.5210 - val_loss: 0.6587 - val_accuracy: 0.2605
Epoch 8/10
21/21 [=====] - ETA: 0s - loss: 0.4562 - accuracy: 0.5150
Epoch 00008: val_loss improved from 0.65866 to 0.65671, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 448ms/step - loss: 0.4562 - accuracy: 0.5150 - val_loss: 0.6567 - val_accuracy: 0.2605
Epoch 9/10
21/21 [=====] - ETA: 0s - loss: 0.4463 - accuracy: 0.4970
Epoch 00009: val_loss improved from 0.65671 to 0.65427, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 458ms/step - loss: 0.4463 - accuracy: 0.4970 - val_loss: 0.6543 - val_accuracy: 0.2605
Epoch 10/10
21/21 [=====] - ETA: 0s - loss: 0.4273 - accuracy: 0.5344
Epoch 00010: val_loss improved from 0.65427 to 0.65038, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 10s 483ms/step - loss: 0.4273 - accuracy: 0.5344 - val_loss: 0.6504 - val_accuracy: 0.2605
```

## 두 번째 모델 예측과 history 그래프

11/11 [=====] - 1s 61ms/step - loss: 0.6505 - accuracy: 0.3018  
정확도 : 0.30



### (3) 세번째 모델

모델의 예측 정확도가 좋지 않아 레이어를 추가하고 에포크를 늘려보았다.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormalization, Activation
from keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt

X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

model = Sequential()
model.add(Conv2D(32, (3,3), padding="same", input_shape=X_train.shape[1:], activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3), padding="same"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization(axis=2))
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3,3), padding="same"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization(axis=2))
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), padding="same"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization(axis=2))
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(4, activation="sigmoid"))

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

model_path = path + "/weather.model"
```

```
checkpoint = ModelCheckpoint(filepath=model_path, monitor='val_loss',
    verbose=1, save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', patience=7)
```

## 세 번째 모델 구조

```
model.summary()
```

Model: "sequential\_7"

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 300, 300, 32)	896
max_pooling2d_29 (MaxPooling2D)	(None, 150, 150, 32)	0
conv2d_31 (Conv2D)	(None, 150, 150, 32)	9248
max_pooling2d_30 (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_32 (Conv2D)	(None, 75, 75, 64)	18496
max_pooling2d_31 (MaxPooling2D)	(None, 37, 37, 64)	0
batch_normalization_3 (Batch Normalization)	(None, 37, 37, 64)	148
activation (Activation)	(None, 37, 37, 64)	0
dropout_19 (Dropout)	(None, 37, 37, 64)	0
conv2d_33 (Conv2D)	(None, 37, 37, 128)	73856
max_pooling2d_32 (MaxPooling2D)	(None, 18, 18, 128)	0
batch_normalization_4 (Batch Normalization)	(None, 18, 18, 128)	72
activation_1 (Activation)	(None, 18, 18, 128)	0
dropout_20 (Dropout)	(None, 18, 18, 128)	0
conv2d_34 (Conv2D)	(None, 18, 18, 256)	295168
max_pooling2d_33 (MaxPooling2D)	(None, 9, 9, 256)	0
batch_normalization_5 (Batch Normalization)	(None, 9, 9, 256)	36
activation_2 (Activation)	(None, 9, 9, 256)	0
dropout_21 (Dropout)	(None, 9, 9, 256)	0
flatten_5 (Flatten)	(None, 20736)	0
dense_10 (Dense)	(None, 512)	10617344
dropout_22 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 4)	2052

=====  
Total params: 11,017,316  
Trainable params: 11,017,188  
Non-trainable params: 128  
=====

## 세 번째 모델 accuracy

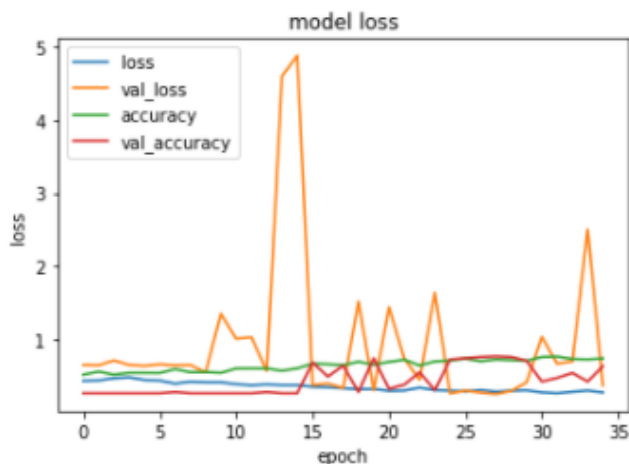
validation loss가 향상되지 않으면 에포크가 남았어도 자동으로 종료한다.

```
Epoch 1/100
21/21 [=====] - ETA: 0s - loss: 0.4293 - accuracy: 0.5150
Epoch 00001: val_loss improved from 0.65038 to 0.64834, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 433ms/step - loss: 0.4293 - accuracy: 0.5150 - val_loss: 0.6483 - val_accuracy: 0.2605
Epoch 2/100
21/21 [=====] - ETA: 0s - loss: 0.4355 - accuracy: 0.5554
Epoch 00002: val_loss improved from 0.64834 to 0.64354, saving model to /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/dataset2/weather.model/assets
21/21 [=====] - 9s 443ms/step - loss: 0.4355 - accuracy: 0.5554 - val_loss: 0.6435 - val_accuracy: 0.2605
Epoch 3/100
21/21 [=====] - ETA: 0s - loss: 0.4656 - accuracy: 0.5150
Epoch 00003: val_loss did not improve from 0.64354
21/21 [=====] - 4s 168ms/step - loss: 0.4656 - accuracy: 0.5150 - val_loss: 0.7095 - val_accuracy: 0.2605
Epoch 4/100
21/21 [=====] - ETA: 0s - loss: 0.4783 - accuracy: 0.5389
Epoch 00004: val_loss did not improve from 0.64354
21/21 [=====] - 4s 172ms/step - loss: 0.4783 - accuracy: 0.5389 - val_loss: 0.6480 - val_accuracy: 0.2605
Epoch 5/100
21/21 [=====] - ETA: 0s - loss: 0.4418 - accuracy: 0.5389

Epoch 31/100
21/21 [=====] - ETA: 0s - loss: 0.2723 - accuracy: 0.7560
Epoch 00031: val_loss did not improve from 0.24932
21/21 [=====] - 4s 169ms/step - loss: 0.2723 - accuracy: 0.7560 - val_loss: 1.0319 - val_accuracy: 0.4202
Epoch 32/100
21/21 [=====] - ETA: 0s - loss: 0.2606 - accuracy: 0.7620
Epoch 00032: val_loss did not improve from 0.24932
21/21 [=====] - 4s 171ms/step - loss: 0.2606 - accuracy: 0.7620 - val_loss: 0.6647 - val_accuracy: 0.4706
Epoch 33/100
21/21 [=====] - ETA: 0s - loss: 0.2810 - accuracy: 0.7290
Epoch 00033: val_loss did not improve from 0.24932
21/21 [=====] - 4s 173ms/step - loss: 0.2810 - accuracy: 0.7290 - val_loss: 0.6967 - val_accuracy: 0.5378
Epoch 34/100
21/21 [=====] - ETA: 0s - loss: 0.2971 - accuracy: 0.7231
Epoch 00034: val_loss did not improve from 0.24932
21/21 [=====] - 3s 167ms/step - loss: 0.2971 - accuracy: 0.7231 - val_loss: 2.5029 - val_accuracy: 0.4202
Epoch 35/100
21/21 [=====] - ETA: 0s - loss: 0.2716 - accuracy: 0.7365
Epoch 00035: val_loss did not improve from 0.24932
21/21 [=====] - 4s 169ms/step - loss: 0.2716 - accuracy: 0.7365 - val_loss: 0.3683 - val_accuracy: 0.6303
```

## 세 번째 모델 예측과 history 그래프

11/11 [=====] - 1s 65ms/step - loss: 0.3479 - accuracy: 0.6864  
정확도 : 0.69



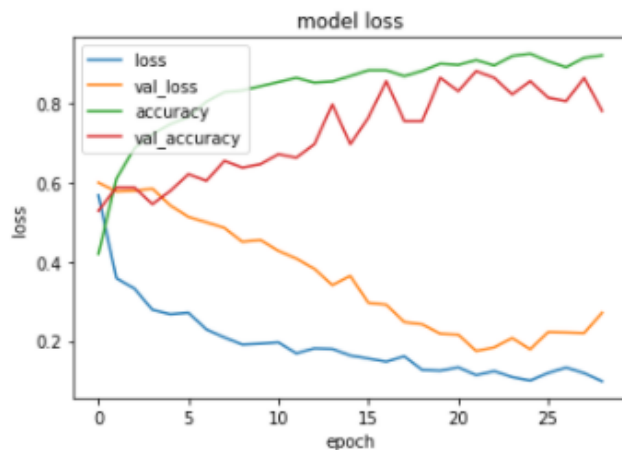
### 3. 이미지 사이즈 별 accuracy 비교

Accuracy를 더 높이기 위한 여러가지 방법을 고안해보았다. 이미지 전처리 시 width와 height를 64로 바꿨더니 accuracy가 증가했다. 머신러닝에서 자주 사용되는 이미지 사이즈는 32X32, 64X64, 96X96, 245X256 라고 한다. (<https://ivo-lee.tistory.com/91>)

```
img_width = 64
img_height = 64
```

```
Epoch 24/100
21/21 [=====] - ETA: 0s - loss: 0.1087 - accuracy: 0.9207
Epoch 00024: val_loss did not improve from 0.17493
21/21 [=====] - 1s 29ms/step - loss: 0.1087 - accuracy: 0.9207 - val_loss: 0.2076 - val_accuracy: 0.8235
Epoch 25/100
21/21 [=====] - ETA: 0s - loss: 0.1001 - accuracy: 0.9266
Epoch 00025: val_loss did not improve from 0.17493
21/21 [=====] - 1s 29ms/step - loss: 0.1001 - accuracy: 0.9266 - val_loss: 0.1791 - val_accuracy: 0.8571
Epoch 26/100
20/21 [=====>..] - ETA: 0s - loss: 0.1220 - accuracy: 0.9031
Epoch 00026: val_loss did not improve from 0.17493
21/21 [=====] - 1s 31ms/step - loss: 0.1194 - accuracy: 0.9072 - val_loss: 0.2230 - val_accuracy: 0.8151
Epoch 27/100
20/21 [=====>..] - ETA: 0s - loss: 0.1334 - accuracy: 0.8906
Epoch 00027: val_loss did not improve from 0.17493
21/21 [=====] - 1s 30ms/step - loss: 0.1332 - accuracy: 0.8922 - val_loss: 0.2220 - val_accuracy: 0.8067
Epoch 28/100
20/21 [=====>..] - ETA: 0s - loss: 0.1171 - accuracy: 0.9156
Epoch 00028: val_loss did not improve from 0.17493
21/21 [=====] - 1s 29ms/step - loss: 0.1188 - accuracy: 0.9162 - val_loss: 0.2203 - val_accuracy: 0.8655
Epoch 29/100
20/21 [=====>..] - ETA: 0s - loss: 0.0984 - accuracy: 0.9219
Epoch 00029: val_loss did not improve from 0.17493
21/21 [=====] - 1s 29ms/step - loss: 0.0983 - accuracy: 0.9222 - val_loss: 0.2717 - val_accuracy: 0.7815
```

```
11/11 [=====] - 0s 19ms/step - loss: 0.2298 - accuracy: 0.8314
정확도 : 0.83
```



예측 정확도가 증가했다.

4의 배수를 쓰는 게 일반적이라고 해서 224로도 맞춰보았다.


```
img_width = 224
img_height = 224
```

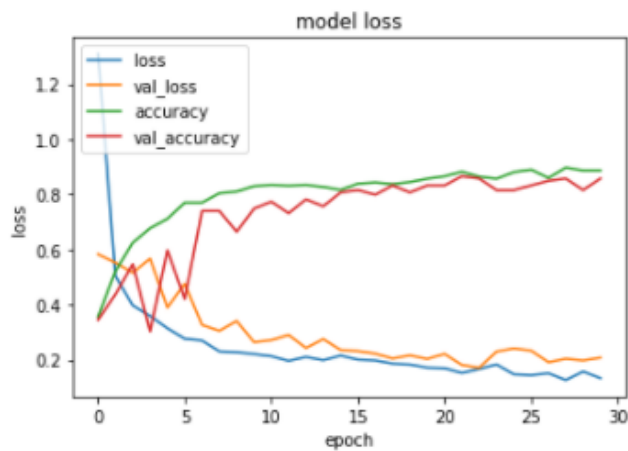
```
✓ [81] print(X_train.shape)
초 (787, 224, 224, 3)
```

```

Epoch 26/100
21/21 [=====] - ETA: 0s - loss: 0.1438 - accuracy: 0.8892
Epoch 00026: val_loss did not improve from 0.17042
21/21 [=====] - 2s 100ms/step - loss: 0.1438 - accuracy: 0.8892 - val_loss: 0.2326 - val_accuracy: 0.8319
Epoch 27/100
21/21 [=====] - ETA: 0s - loss: 0.1507 - accuracy: 0.8593
Epoch 00027: val_loss did not improve from 0.17042
21/21 [=====] - 2s 102ms/step - loss: 0.1507 - accuracy: 0.8593 - val_loss: 0.1906 - val_accuracy: 0.8487
Epoch 28/100
21/21 [=====] - ETA: 0s - loss: 0.1258 - accuracy: 0.8967
Epoch 00028: val_loss did not improve from 0.17042
21/21 [=====] - 2s 100ms/step - loss: 0.1258 - accuracy: 0.8967 - val_loss: 0.2032 - val_accuracy: 0.8571
Epoch 29/100
21/21 [=====] - ETA: 0s - loss: 0.1569 - accuracy: 0.8862
Epoch 00029: val_loss did not improve from 0.17042
21/21 [=====] - 2s 100ms/step - loss: 0.1569 - accuracy: 0.8862 - val_loss: 0.1970 - val_accuracy: 0.8151
Epoch 30/100
21/21 [=====] - ETA: 0s - loss: 0.1328 - accuracy: 0.8862
Epoch 00030: val_loss did not improve from 0.17042
21/21 [=====] - 2s 100ms/step - loss: 0.1328 - accuracy: 0.8862 - val_loss: 0.2073 - val_accuracy: 0.8571

```

 11/11 [=====] - 0s 41ms/step - loss: 0.2119 - accuracy: 0.8905  
 정확도 : 0.89



예측 정확도가 증가하였다.

#### 4. 소감

그동안 만들어보았던 모델은 숫자로 된 데이터를 가지고 예측을 하는 모델이었다. 이미지를 가지고 예측모델을 만드려니 막막해서 구글링을 통해 괜찮은 예시 소스들을 찾아보았다.

이미지를 사용하기 위해 ImageDataGenerator를 통한 전처리와 Image 객체를 이용한 전처리 두 가지 중 전자를 선택했다가 오류가 많이 발생해서 후자로 바꿨다. 다음에는 ImageDataGenerator를 통해서 전처리를 진행해보고 싶다.

데이터 셋의 양이 많지 않아 매우 높은 정확도를 기대하기는 어려웠다. 사진의 각도를 바꿔서 데이터 양을 늘려볼까 했지만 날씨 데이터는 하늘과 땅의 위치가 바뀌는 일이 생기면 안될 것 같아서 그렇게 데이터 양을 늘리지는 않기로 했다.

그래도 에포크를 늘리니 accuracy가 눈에 띄게 증가하는 것을 볼 수 있었다. 숫자로 된 데이터보다 accuracy 증가율은 더 큰 것 같았다.

colab에서 모델을 학습시켰는데 CPU로 학습하는 것보다 GPU로 학습하는 것이 훨씬 빠르다는 것을 알게 되었다. colab 기본 설정이 GPU를 사용하도록 돼있는 줄 알았는데 아니었던 것이다. 다음부터는 GPU를 이용할 것이다.

딥러닝의 큰 장점이 이미지를 분류할 수 있는 점인 것 같은데, 이번 기회를 통해서 이미지 분류 모델을 만들어볼 수 있어 좋았다. 한 학기동안의 수업의 결실을 맺은 과제였던 것 같다.