딥러닝/클라우드-기말고사대체과제

딥러닝/클라우드 1분반
32170578 김산

1. Library Import

gpu가속을 활용하기 위해 google colab을 활용하였습니다.

colab에서는 Nvidia K80 GPU를 제공하여 고성능의 GPU 가속을 사용할 수 있어 많은 양의 이미지를 원활하게 학습시킬 수 있었습니다.

제 구글 drive에 업로드한 데이터를 가져오기위해 drive를 마운트 합니다.

```
from google.colab import drive
drive.mount("/content/drive/")
```

split-folders
split-folders라이브러리는 train, val, test의 폴더트리로 나누어 훈련 데이터 셋과 테스트 데이터 셋을 만들어 줍니다.

```
!pip install split-folders
```

```
import os
import shutil
import splitfolders
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.layers import Dense, Activation, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

2. Classify Image by folders

```
# 이미지를 폴더별로 분류하여 저장
def directory_clf(path_dir, copy_path_dir):
    img_list = os.listdir(path_dir)
    weather = ['shine','sunrise', 'cloudy', 'rain']

    # 각 파일에 포함된 글자를 통해 이미지 dictionary 분류
    img_dict = {}
    for i in weather:
        img_dict[i] = [x for x in img_list if i in x]

    # 디렉토리가 없을 경우 생성
    if not os.path.exists(copy_path_dir):
        os.makedirs(copy_path_dir)

    # 각 디렉토리로 분류하여 복사
    for i in weather:
        if not os.path.exists(copy_path_dir + '/' + i):
            os.makedirs(copy_path_dir + '/' + i)

        for file_name in img_dict[i]:
            origin_path = path_dir + '/' + file_name
            copy_path   = copy_path_dir + '/' + i + '/' + file_name
            shutil.copy(origin_path, copy_path)
```

```
path_dir = "./drive/MyDrive/weather_dataset"
copy_path_dir = "./drive/MyDrive/weather_dataset_clf"
train_val_dir_path = "./drive/MyDrive/weather_train_val"
```

```
directory_clf(path_dir, copy_path_dir)
```

각 클래스별 이미지를 7 : 3 으로 나눕니다

```
splitfolders.ratio(copy_path_dir, output=train_val_dir_path, seed=123, ratio=
(0.7, 0.3))
```

```
(train, val) = (0, 0)
for root, dirs, files in os.walk(train_val_dir_path + "/train"):
    train += len(files)
print('Train set size: ',train)

for root, dirs, files in os.walk(train_val_dir_path + "/val"):
    val += len(files)
print('Test set size: ',val)
```

```
Train set size:  786
Test set size:  339
```

3. Image Generator

ImageDataGenerator를 통해 학습데이터를 조금씩 변형시켜 학습 데이터의 양을 늘렸습니다.

텐서플로우의 공식문서 https://www.tensorflow.org/tutorials/images/data_augmentation?hl=ko 를 참고하면 데이터 증강은 훈련세트에만 적용해야 한다 라고 명시되어 있으므로 훈련세트에만 적용하였습니다.

또한 이미지의 크기가 다양한 크기를 가지고 있기때문에 모든 추출 이미지를 고정된 크기로 조정하였습니다.

```
train_data_dir = train_val_dir_path + '/train'
val_data_dir = train_val_dir_path + '/val'

target_size = (256, 256)
batch_size = 32

train_datagen = ImageDataGenerator(
    rescale = 1./255,                # 값을 0과 1 사이로 변경
    shear_range = 0.2,               # 층 밀리기 강도
    zoom_range = 0.2,                # 줌 범위
    horizontal_flip = True,          # 무작위 가로 뒤집기
    width_shift_range=0.2,           # 가로 이동
    height_shift_range=0.2,          # 세로 이동
    rotation_range=20,               # 회전
    brightness_range=[0.5, 1.5]      # 밝기 범위
)

test_datagen = ImageDataGenerator(
    rescale = 1./255
)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size = target_size,
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle=True
)

test_generator = test_datagen.flow_from_directory(
    val_data_dir,
    target_size = target_size,
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle=True
)
```

```
Found 786 images belonging to 4 classes.
Found 339 images belonging to 4 classes.
```

4. Build Model

1. Model 1

첫번째 모델은 convolution layer와 maxpooling layer를 통해 간단한 CNN모델을 만들어 실험하였습니다.

- Maxpooling layer : 이미지에서 일부 픽셀을 제거하여 이미지의 차수를 감소시킵니다. MaxPooling(최대 풀링)은 이미지의 n x n 영역을 해당 영역의 최대 픽셀 값으로 대체하여 이미지를 다운샘플링합니다.
- Flatten : dense layer로의 전환을 위해 입력된 특징 맵의 모든 화소를 나열하여 하나의 벡터로 평탄화 작업을 수행합니다.

```
img_dims = (250, 250, 3)

def build_model_1(width, height, depth) :

    input_shape = (height, width, depth)

    model = models.Sequential()

    model.add(Conv2D(32, kernel_size=(5, 5),strides=(1, 1), padding='same',
                     activation='relu', input_shape=input_shape))

    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Flatten())
    model.add(Dense(64,activation='relu'))
    model.add(Dense(4,activation='softmax'))

    return model

model_1 = build_model_1(img_dims[0], img_dims[1], img_dims[2])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 250, 250, 32)      2432

 max_pooling2d (MaxPooling2D  (None, 125, 125, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        36928

 flatten (Flatten)           (None, 230400)            0

 dense (Dense)               (None, 64)                14745664

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 14,803,780
Trainable params: 14,803,780
Non-trainable params: 0
_________________________________________________________________
```

```
def compile_train_model(model, epoch, batch_size):

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

    #Training the model
    history = model.fit(
        train_generator,
        batch_size = batch_size,
        validation_data = test_generator,
        epochs = epoch,
    )
    return history
```

```
hist_1 = compile_train_model(model_1, 10, 128)
```

Training last 5 epoch - 1

```
Epoch 5/10
25/25 [==============================] - 18s 752ms/step - loss: 0.5935 - accuracy: 0.7952 - val_loss: 0.4897 - val_accuracy: 0.8407
Epoch 6/10
25/25 [==============================] - 18s 716ms/step - loss: 0.4289 - accuracy: 0.8448 - val_loss: 0.3791 - val_accuracy: 0.8879
Epoch 7/10
25/25 [==============================] - 18s 698ms/step - loss: 0.5087 - accuracy: 0.8092 - val_loss: 0.5425 - val_accuracy: 0.8142
Epoch 8/10
25/25 [==============================] - 18s 733ms/step - loss: 0.4750 - accuracy: 0.8155 - val_loss: 0.4329 - val_accuracy: 0.8702
Epoch 9/10
25/25 [==============================] - 18s 733ms/step - loss: 0.3764 - accuracy: 0.8626 - val_loss: 0.3696 - val_accuracy: 0.8909
Epoch 10/10
25/25 [==============================] - 18s 709ms/step - loss: 0.3831 - accuracy: 0.8588 - val_loss: 0.3337 - val_accuracy: 0.9027
```

Test loss, Test accuracy - 1

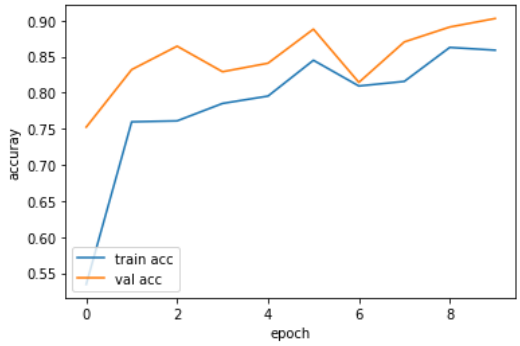
```
def print_accuracy(model):
    scores = model.evaluate(test_generator, steps=5)
    print("Test loss: ", scores[0])
    print("Test accuracy: ", scores[1])

print_accuracy(model_1)
```

```
5/5 [==============================] - 1s 194ms/step - loss: 0.2586 - accuracy: 0.9000
Test loss:  0.25861525535583496
Test accuracy:  0.8999999761581421
```

학습 곡선 그래프 - 1

```
def train_curve_graph(history):
    fig, acc_ax = plt.subplots()

    acc_ax.plot(history.history['accuracy'], label='train acc')
    acc_ax.plot(history.history['val_accuracy'], label='val acc')

    acc_ax.set_xlabel('epoch')
    acc_ax.set_ylabel('accuray')
```

```
    acc_ax.legend(loc='lower left')

    plt.show()

train_curve_graph(hist_1)
```



batch_size를 128로 하여 학습을 진행하였는데 size가 너무 커서인지 학습이 진행됨에 따라 정확도가 튀는 모습을 확인할 수 있었습니다. 따라서 batch_size를 16으로 낮추어 학습을 다시 진행하였습니다.

```
model_1 = build_model_1(img_dims[0], img_dims[1], img_dims[2])
hist_1 = compile_train_model(model_1, 10, 16)
```

Training last 5 epoch - 2

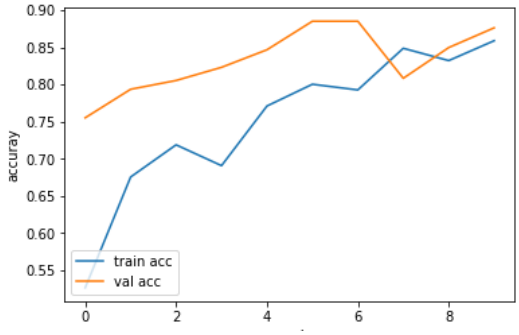
```
Epoch 6/10
25/25 [==============================] - 18s 716ms/step - loss: 0.5126 - accuracy: 0.8003 - val_loss: 0.3893 - val_accuracy: 0.8850
Epoch 7/10
25/25 [==============================] - 18s 717ms/step - loss: 0.5117 - accuracy: 0.7926 - val_loss: 0.4281 - val_accuracy: 0.8850
Epoch 8/10
25/25 [==============================] - 18s 714ms/step - loss: 0.4376 - accuracy: 0.8486 - val_loss: 0.5669 - val_accuracy: 0.8083
Epoch 9/10
25/25 [==============================] - 18s 712ms/step - loss: 0.4340 - accuracy: 0.8321 - val_loss: 0.4570 - val_accuracy: 0.8496
Epoch 10/10
25/25 [==============================] - 18s 704ms/step - loss: 0.4019 - accuracy: 0.8588 - val_loss: 0.3994 - val_accuracy: 0.8761
```

```
print_accuracy(model_1)
```

Test loss, Test accuracy - 2

```
5/5 [==============================] - 1s 171ms/step - loss: 0.3700 - accuracy: 0.8687
Test loss:  0.3699573576450348
Test accuracy:  0.8687499761581421
```

학습 곡선 그래프 - 2



그래프를 확인하면 계속해서 accuracy가 증가함을 확인할 수 있습니다. 적절한 epoch을 찾기 위해 epoch을 늘려 다시 학습을 진행하였습니다.

Training last 5 epoch - 3

```
model_1 = build_model_1(img_dims[0], img_dims[1], img_dims[2])
hist_1 = compile_train_model(model_1, 30, 16)
```

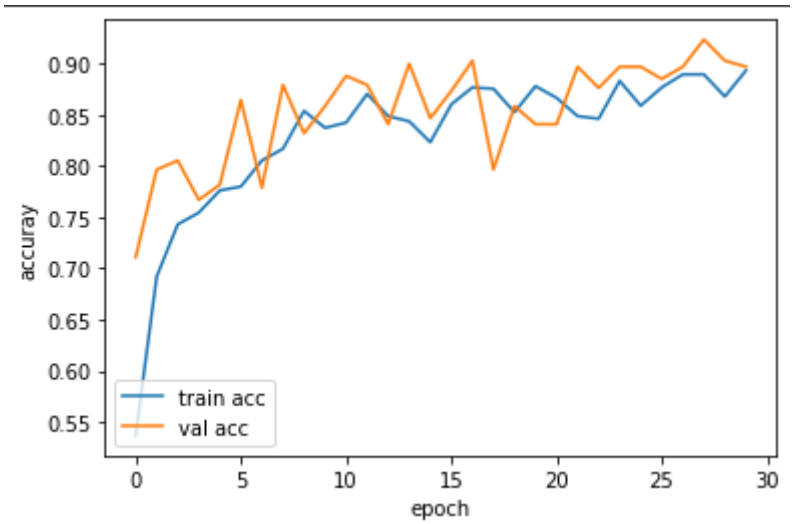
```
Epoch 26/30
25/25 [==============================] - 18s 707ms/step - loss: 0.2837 - accuracy: 0.8766 - val_loss: 0.4268 - val_accuracy: 0.8850
Epoch 27/30
25/25 [==============================] - 18s 713ms/step - loss: 0.2828 - accuracy: 0.8893 - val_loss: 0.3991 - val_accuracy: 0.8968
Epoch 28/30
25/25 [==============================] - 18s 703ms/step - loss: 0.2970 - accuracy: 0.8893 - val_loss: 0.3349 - val_accuracy: 0.9233
Epoch 29/30
25/25 [==============================] - 18s 696ms/step - loss: 0.3176 - accuracy: 0.8677 - val_loss: 0.3451 - val_accuracy: 0.9027
Epoch 30/30
25/25 [==============================] - 18s 702ms/step - loss: 0.2645 - accuracy: 0.8931 - val_loss: 0.3736 - val_accuracy: 0.8968
```

Test loss, Test accuracy - 3

```
print_accuracy(model_1)
```

학습 곡선 그래프 - 3

```
train_curve_graph(hist_1)
```



학습이 진행함에 따라 정확도가 더이상 증가하지 않고 계속 88%대에서 머물고 있음을 확인할 수 있습니다. 정확도를 좀더 높이기 위해 layer수를 추가한 Model을 구현하여 학습을 진행하였습니다

2. Model 2

```
def build_model_2(width, height, depth) :

    input_shape = (height, width, depth)

    model = models.Sequential()

    model.add(Conv2D(32, kernel_size=(5, 5),strides=(1, 1), padding='same',
                     activation='relu', input_shape=input_shape))

    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
```

```
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(512, (3, 3), activation='relu'))

    model.add(Flatten())
    model.add(Dense(512,activation='relu'))
    model.add(Dense(256,activation='relu'))
    model.add(Dense(128,activation='relu'))
    model.add(Dense(4,activation='softmax'))

    return model

model_2 = build_model_2(img_dims[0], img_dims[1], img_dims[2])
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 250, 250, 32)      2432

 max_pooling2d_18 (MaxPoolin  (None, 125, 125, 32)     0
 g2D)

 conv2d_25 (Conv2D)          (None, 125, 125, 64)      18496

 max_pooling2d_19 (MaxPoolin  (None, 62, 62, 64)       0
 g2D)

 conv2d_26 (Conv2D)          (None, 62, 62, 128)       73856

 max_pooling2d_20 (MaxPoolin  (None, 31, 31, 128)      0
 g2D)

 conv2d_27 (Conv2D)          (None, 31, 31, 256)       295168

 max_pooling2d_21 (MaxPoolin  (None, 15, 15, 256)      0
 g2D)

 conv2d_28 (Conv2D)          (None, 15, 15, 512)       1180160

 max_pooling2d_22 (MaxPoolin  (None, 7, 7, 512)        0
 g2D)

 conv2d_29 (Conv2D)          (None, 5, 5, 512)         2359808

 flatten_6 (Flatten)         (None, 12800)             0

 dense_16 (Dense)            (None, 512)               6554112

 dense_17 (Dense)            (None, 256)               131328

 dense_18 (Dense)            (None, 128)               32896

 dense_19 (Dense)            (None, 4)                 516

=================================================================
Total params: 10,648,772
Trainable params: 10,648,772
Non-trainable params: 0
_________________________________________________________________
```

Training last 5 epoch

```
Epoch 6/10
25/25 [==============================] - 18s 707ms/step - loss: 0.6670 - accuracy: 0.6908 - val_loss: 0.6250 - val_accuracy: 0.7434
Epoch 7/10
25/25 [==============================] - 18s 704ms/step - loss: 0.6281 - accuracy: 0.7188 - val_loss: 0.5223 - val_accuracy: 0.8112
Epoch 8/10
25/25 [==============================] - 18s 710ms/step - loss: 0.6593 - accuracy: 0.6896 - val_loss: 0.5975 - val_accuracy: 0.7286
Epoch 9/10
25/25 [==============================] - 18s 711ms/step - loss: 0.6598 - accuracy: 0.7087 - val_loss: 0.6347 - val_accuracy: 0.7139
Epoch 10/10
25/25 [==============================] - 18s 711ms/step - loss: 0.6605 - accuracy: 0.7239 - val_loss: 0.8469 - val_accuracy: 0.6077
```

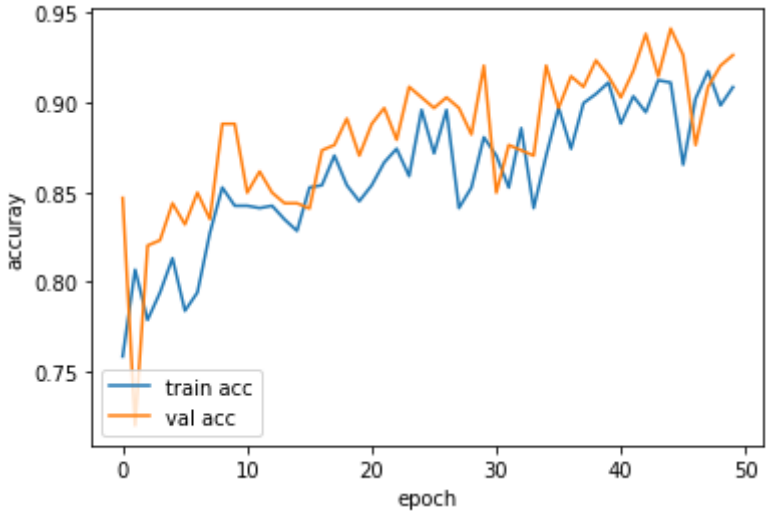
ecoch을 기존과 같이 10으로 지정하여 학습을 진행하였을 때는 layer수가 많아 학습이 제대로 되지않는 것을 확인할 수 있었습니다. epoch수를 50으로 늘려 학습을 더 진행하였습니다.

```
Epoch 46/50
25/25 [==============================] - 18s 712ms/step - loss: 0.3497 - accuracy: 0.8651 - val_loss: 0.3203 - val_accuracy: 0.9263
Epoch 47/50
25/25 [==============================] - 18s 705ms/step - loss: 0.2677 - accuracy: 0.9020 - val_loss: 0.3329 - val_accuracy: 0.8761
Epoch 48/50
25/25 [==============================] - 18s 712ms/step - loss: 0.1983 - accuracy: 0.9173 - val_loss: 0.3884 - val_accuracy: 0.9086
Epoch 49/50
25/25 [==============================] - 18s 706ms/step - loss: 0.2947 - accuracy: 0.8982 - val_loss: 0.2507 - val_accuracy: 0.9204
Epoch 50/50
25/25 [==============================] - 18s 710ms/step - loss: 0.2196 - accuracy: 0.9084 - val_loss: 0.3040 - val_accuracy: 0.9263
```

Test loss, Test accuracy

```
5/5 [==============================] - 1s 168ms/step - loss: 0.3244 - accuracy: 0.9125
Test loss:  0.3244237005710602
Test accuracy:  0.9125000238418579
```

학습 곡선 그래프



test정확도가 model 1의 87%에 비해 model2는 91.25%로 증가한것을 확인할 수 있습니다.

3. Model 3

Model 2에서 layer수를 늘림에 따라 정확도가 상승한 것을 확인할 수 있었습니다. 이에 layer수를 많이 늘려 학습을 진행을 하였습니다.

layer수가 많아지게 되면 같은 Input 값을 갖더라도, 가중치가 조금만 달라지면 완전히 다른 값을 얻을 수 있게됩니다. 이를 해결하기 위해, 각 layer에 배치 정규화 layer를 추가하여, 가중치의 차이를 완화할 수 있도록 하였습니다.

또 과적합 문제를 방지하기 위해 dropout layer를 추가하였습니다.

```
def build_model_3(width, height, depth) :

    input_shape = (height, width, depth)

    model = models.Sequential()

    model.add(Conv2D(32, kernel_size=(5, 5),strides=(1, 1), padding='same',
                     input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Activation("relu"))

    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation("relu"))

    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation("relu"))

    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation("relu"))

    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation("relu"))

    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation("relu"))

    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation("relu"))

    model.add(MaxPooling2D((2, 2)))

    model.add(GlobalAveragePooling2D())

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

    model.add(Dense(128, activation = "relu"))
    model.add(Dropout(0.5))

    model.add(Dense(4, activation = 'softmax'))

    return model

model_3 = build_model_3(img_dims[0], img_dims[1], img_dims[2])
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_39 (Conv2D)          (None, 250, 250, 32)      2432

 batch_normalization_8 (Batc  (None, 250, 250, 32)     128
 hNormalization)

 activation_7 (Activation)   (None, 250, 250, 32)      0

 conv2d_40 (Conv2D)          (None, 250, 250, 32)      9248

 batch_normalization_9 (Batc  (None, 250, 250, 32)     128
 hNormalization)

 max_pooling2d_27 (MaxPoolin  (None, 125, 125, 32)     0
 g2D)

 conv2d_41 (Conv2D)          (None, 125, 125, 64)      18496

 batch_normalization_10 (Bat  (None, 125, 125, 64)     256
 chNormalization)

 activation_8 (Activation)   (None, 125, 125, 64)      0

 conv2d_42 (Conv2D)          (None, 125, 125, 64)      36928

 batch_normalization_11 (Bat  (None, 125, 125, 64)     256
 chNormalization)

 activation_9 (Activation)   (None, 125, 125, 64)      0

 max_pooling2d_28 (MaxPoolin  (None, 62, 62, 64)       0
 g2D)

 conv2d_43 (Conv2D)          (None, 62, 62, 128)       73856

 batch_normalization_12 (Bat  (None, 62, 62, 128)      512
 chNormalization)

 activation_10 (Activation)  (None, 62, 62, 128)       0

 conv2d_44 (Conv2D)          (None, 62, 62, 128)       147584

 batch_normalization_13 (Bat  (None, 62, 62, 128)      512
 chNormalization)

 activation_11 (Activation)  (None, 62, 62, 128)       0

 max_pooling2d_29 (MaxPoolin  (None, 31, 31, 128)      0
 g2D)

 conv2d_45 (Conv2D)          (None, 31, 31, 256)       295168

 batch_normalization_14 (Bat  (None, 31, 31, 256)      1024
 chNormalization)
 activation_12 (Activation)  (None, 31, 31, 256)       0

 conv2d_46 (Conv2D)          (None, 31, 31, 256)       590080

 batch_normalization_15 (Bat  (None, 31, 31, 256)      1024
 chNormalization)

 activation_13 (Activation)  (None, 31, 31, 256)       0

 max_pooling2d_30 (MaxPoolin  (None, 15, 15, 256)      0
 g2D)

 global_average_pooling2d (G  (None, 256)              0
 lobalAveragePooling2D)

 dense_20 (Dense)            (None, 512)               131584

 dropout (Dropout)           (None, 512)               0

 dense_21 (Dense)            (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_22 (Dense)            (None, 4)                 516

=================================================================
Total params: 1,375,396
Trainable params: 1,373,476
Non-trainable params: 1,920
_________________________________________________________________
```

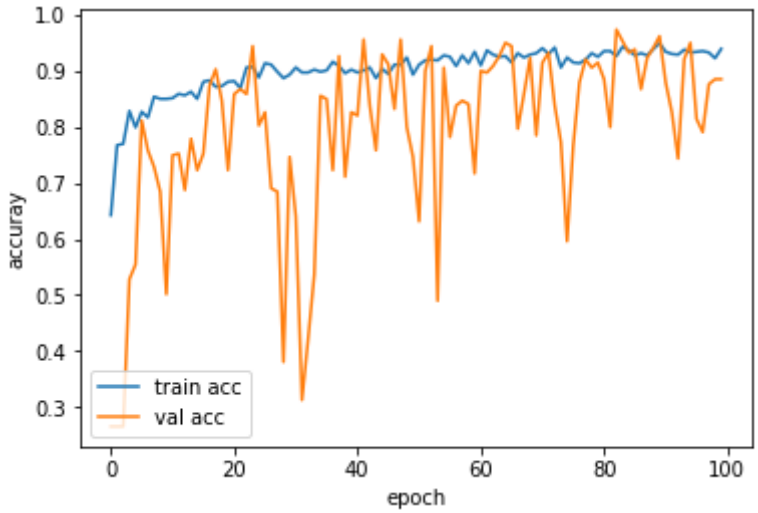
Training Last 5 epoch - 1

```
Epoch 96/100
25/25 [==============================] - 18s 724ms/step - loss: 0.1710 - accuracy: 0.9338 - val_loss: 0.6042 - val_accuracy: 0.8142
Epoch 97/100
25/25 [==============================] - 18s 719ms/step - loss: 0.1713 - accuracy: 0.9351 - val_loss: 0.5046 - val_accuracy: 0.7906
Epoch 98/100
25/25 [==============================] - 18s 712ms/step - loss: 0.1747 - accuracy: 0.9326 - val_loss: 0.3369 - val_accuracy: 0.8761
Epoch 99/100
25/25 [==============================] - 18s 713ms/step - loss: 0.1940 - accuracy: 0.9224 - val_loss: 0.4157 - val_accuracy: 0.8850
Epoch 100/100
25/25 [==============================] - 18s 719ms/step - loss: 0.1675 - accuracy: 0.9389 - val_loss: 0.3166 - val_accuracy: 0.8850
```

Test loss, Test accuracy - 1

```
5/5 [==============================] - 1s 204ms/step - loss: 0.3226 - accuracy: 0.8875
Test loss:  0.32255420088768005
Test accuracy:  0.887499988079071
```

학습 곡선 그래프 - 1



Model 3의 학습 곡선을 확인하면 정확도가 증가하지만, 학습이 제대로 되었다고 보기 힘들 정도로 Validation set의 정확도가 심하게 왔다갔다하는것을 확인할 수 있습니다. 이에 모델 fit단계에서 callback함수를 적용하여 보았습니다.

적용한 Callback함수

```
def callbacks():
    lr = ReduceLROnPlateau(
        monitor='val_accuracy',
        factor=0.1,
        patience=5,
        min_lr=1e-30,
        cooldown=3
    )
    return [lr]

def compile_train_model_2(model, epoch, batch_size):

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

    #Training the model
```

```
    history = model.fit(
        train_generator,
        batch_size = batch_size,
        validation_data = test_generator,
        epochs = epoch,
        callbacks=callbacks()
    )
    return history
```

정확도가 발산하는 문제를 해결하기 위해 val accuracy를 기준으로 정확도 향상이 없을시 learning rate를 감소시켜 보았습니다.

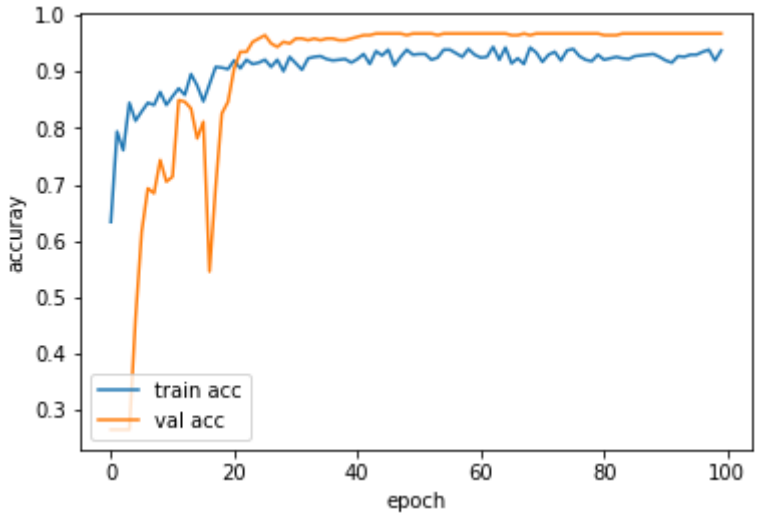
Training Last 5 epoch - 2

```
Epoch 96/100
25/25 [==============================] - 18s 715ms/step - loss: 0.2118 - accuracy: 0.9300 - val_loss: 0.1528 - val_accuracy: 0.9676 - lr: 1.0000e-13
Epoch 97/100
25/25 [==============================] - 18s 720ms/step - loss: 0.1886 - accuracy: 0.9351 - val_loss: 0.1530 - val_accuracy: 0.9676 - lr: 1.0000e-13
Epoch 98/100
25/25 [==============================] - 18s 720ms/step - loss: 0.1990 - accuracy: 0.9389 - val_loss: 0.1534 - val_accuracy: 0.9676 - lr: 1.0000e-13
Epoch 99/100
25/25 [==============================] - 18s 716ms/step - loss: 0.2269 - accuracy: 0.9198 - val_loss: 0.1530 - val_accuracy: 0.9676 - lr: 1.0000e-14
Epoch 100/100
25/25 [==============================] - 18s 713ms/step - loss: 0.1843 - accuracy: 0.9377 - val_loss: 0.1532 - val_accuracy: 0.9676 - lr: 1.0000e-14
```

Test loss, Test accuracy - 2

```
5/5 [==============================] - 1s 159ms/step - loss: 0.1517 - accuracy: 0.9812
Test loss:  0.15169189870357513
Test accuracy:  0.981249988079071
```

학습 곡선 그래프 - 2



5. 소감

우선 파일명에 이미지의 class에 대한 내용이 담겨있으므로 이미지를 다운로드 받은 폴더의 경로에 대해 `listdir()` 함수를 사용하여 파일 명을 불러오고 class에 따라 폴더를 생성하여 분류하였습니다. 이후 split-folder 라이브러리를 통해 요구사항에 따라 train, test를 각 7:3의 비율로 나누었습니다.

훈련 데이터가 충분하지 않다고 생각해 Image_generator를 통해 훈련 이미지 개수를 증강시켰습니다. 또한 이미지의 크기가 제각각이므로 Image_generator의 target_size 매개변수를 통해 이미지의 크기를 조정하였습니다.

모델의 경우 간단한 CNN신경망을 먼저 구현한 후 이를 기준으로 정확도를 높여가는 방식으로 더 정확한 모델을 찾으려 노력했습니다. 이미지 데이터가 많다보니 학습속도가 느릴것 같아 batch_size를 128로 크게 잡아 학습을 진행하였습니다. 때문에 학습이 진행됨에 따라 정확도가 튀는 현상이 발생하여 batch_size를 16으로 작게 잡아 학습을 진행하였습니다.

학습의 속도가 느린편이라 Google Colab을 사용하였고, GPU를 사용해 학습을 진행할 수 있도록 Colab의 리소스 세팅을 조정하였습니다.

정확도를 높이기 위해 layer수를 늘려 학습을 진행하였는데 기존에는 10 epoch으로도 충분히 학습이 진행되었으나 layer수가 많아질수록 10 epoch으로도 충분하지 않았습니다. 이에따라 epoch수를 늘려 학습을 진행하였습니다.

정확도가 올라간것을 확인하고 더 높은 정확도를 위해 layer수를 크게 늘렸습니다. layer수가 많아지게 되면 같은 Input 값을 갖더라도, 가중치가 조금만 달라지면 완전히 다른 값을 얻을 수 있게됩니다. 이를 해결하기 위해, 각 layer에 배치 정규화 layer를 추가하여, 가중치의 차이를 완화할 수 있도록 하였습니다. 또한 과적합 문제를 방지하기 위해 dropout layer를 추가하였습니다.

100 epoch으로 진행하였는데 오히려 정확도가 떨어지고, 학습곡선 그래프를 통해 튀는현상이 크게 증가한것을 볼 수 있었습니다. 이를 해결하기 위해 학습이 진행됨에 따라 `val_accuracy`값의 개선이 없으면 learning rate를 감소시키는 `ReduceLROnPlateau()`를 콜백함수로 적용하여 학습을 진행하였습니다.

튀는현상이 크게 줄어들었고 정확도를 크게 향상시킬 수 있었습니다.

평소에 CNN의 등장으로 영상 센서가 다양한 종류의 센서를 대체할 수 있게 되었다고 생각해 CNN알고리즘에 관심이 많았습니다. 이번 대체과제를 통해 CNN알고리즘에 대해 많이 공부할 시간을 가질 수 있어 즐거웠습니다.

0.1% 정확도를 높이는게 큰 의미가 있는지 의문이 들기도 했습니다. 과제를 하면서 딥러닝 분야에서 실무를 하고 계신 분의 글을 읽은적이 있는데 0.1%의 정확도 향상이라도 사용자가 많은 환경에서는 엄청난 수익차이가 발생할 수 있다는 내용이었습니다.

이번학기 딥러닝/클라우드 과목을 들으며 데이터 전처리의 중요성과, 기계학습, 딥러닝 모델들의 특징을 배우면서 인공지능이라는 도메인을 조금이나마 이해할 수 있었고, 여러분야에 써먹을 수 있는 무기가 생긴것 같아 좋았습니다. 감사합니다.