

인공지능 학기말 프로젝트  
(텐서플로 딥러닝<꽃 분류>)

진민주

202001834

컴퓨터공학과

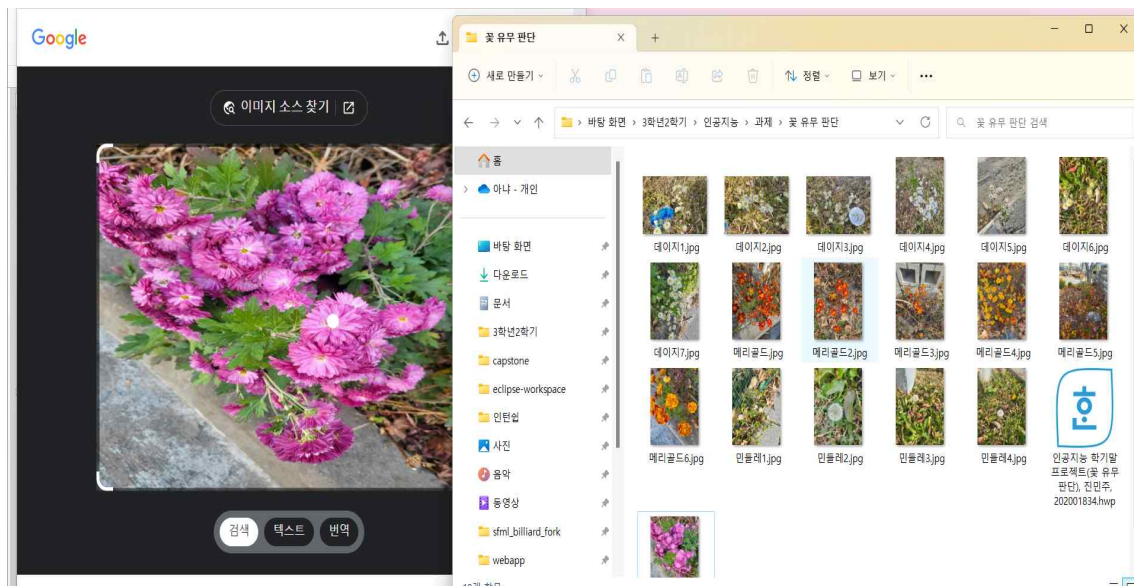
## 1. 서론

### 1.1 꽃 종류 결정 및 test 데이터 구하기

이번 과제는 꽃 분류를 하는 과제인데 제일 큰 문제점이 있었다. train은 크롤링으로 하거나 하면 될 것 같지만 test 데이터는 직접 찍은 사진이어야 했다.

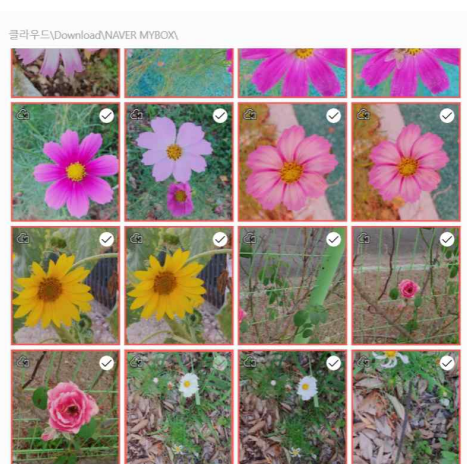
그래서 우선은 겨울이 거의 되었다 보니 최대한 주변에 보이는 꽃들을 무작정 찍었다.  
(8공 앞 꽃, 학교 후문에 있는 꽃 등)

막상 찍고 보니까 어떤 꽃인지 몰라 google 이미지 검색으로 이름을 알아냈다.



찍은 사진들의 종류는 데이지, 민들레(홀씨), 메리골드, 참취속으로 총 4가지이다. 근데 민들레는 홀씨만 있거나 참취속이 한 장만 있거나 애매했다.

고등학교 때 꽃 사진을 자주 찍었던 기억이 나서 19년도 사진을 찾아서 포함했다.



그래서 class 종류는 test 사진 수 3개 이상으로 있는 데이지(daisy), 메리골드(marigold), 접시꽃(hollyhock), 코스모스(cosmos)로 결정했다. test set은 비슷한 사진들 제외하고

class 별로 3장 총 12장으로 진행했다.

## 1.2 train 데이터 구하기

daisy는 kaggle에서 가져올 수 있지만 나머지 사진들은 kaggle에 있어도 해당 꽃 사진이 아닌 것도 섞여 있었다. 그래서 찾아보니 크롬 확장프로그램 image downloader를 이용해서 다운로드 받는 방법이 있어 해당 프로그램을 사용했다. 데이터 수는 많을수록 좋으니 구할 수 있을 만큼 구했다. 이상한 데이터들 삭제하고 나니까 대략 한 class당 700장~800장 가까이 모았다.

<https://www.kaggle.com/datasets/alxmamaev/flowers-recognition>→daisy 다운로드

## 2. 본론

### 2.1 이미지 전처리 (코드 설명)

우선 이미지들의 사이즈가 모두 제각각이었다. 그리고 VGG 사전학습 모델을 이용할 것이기 때문에 처음부터 224x224 사이즈로 아래의 코드를 사용해서 맞췄다.

```
import os
import glob
from PIL import Image

files = glob.glob(
    r"C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류/train_data/*/*.jpg")

for f in files:
    img = Image.open(f)
    img_resize = img.resize((224, 224))
    title, ext = os.path.splitext(f)
    img_resize.save(title + ext)
```

그리고 class별 폴더 사진들 이름도 아래의 코드로 정리했다. 정리할 꽃 이름이 marigold가 아니면 색칠한 자리들에 다른 꽃 이름을 넣으면 되는 코드이다.

```
import os
file_path = r"C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/test_data/marigold"
file_name = os.listdir(file_path)

i = 1 # 파일에 들어가는 숫자
cnt = 1 # 돌아가는 횟수
for name in file_name:
```

```

src = os.path.join(file_path, name)
path, ext = os.path.splitext(name)

dst = 'marigold' + str(i) + ext
print(dst)
dst = os.path.join(file_path, dst)
os.rename(src, dst)
cnt += 1
i += 1

```

## 2.2 이미지 로드 및 전처리 (train, test 코드의 이미지 전처리 부분 설명)

-> 이후 실험을 하다가 train 해보고 괜찮은 weights를 test에 돌리기 위해서 train하는 코드, test하는 코드를 나눴다. 그래서 2.3에서 train 코드 설명, 3.1에서 test 코드 설명이 있는데 그 부분들에선 해당 2.2 이미지 로드 및 전처리 부분은 제외했다. (최종 코드는 부록)

자주 사용하게 될 224 사이즈를 변수로 저장했다. 그리고 수집한 영상들을 불러오고, list에 저장했다. 그리고 train\_img들을 섞었다.

```

#사이즈 지정
img_size = 224

#train, test path
train_path = pathlib.Path('C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류/train_data')
test_path = pathlib.Path('C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류/test_data')

#이미지 list에 저장
train_img = list(train_path.glob('*/*.jpg'))
test_img = list(test_path.glob('*/*.jpg'))

random.shuffle(train_img)

```

영상을 숫자로 받아올 변수를 만들고, class를 분류하기 위해서 cosmos는 0, daisy는 1, hollyhock는 2, marigold는 3으로 변수를 지정했다.

```

#영상 읽어오기
x_train, x_test = [], []
y_train, y_test = [], []
y = {'cosmos': 0, 'daisy': 1, 'hollyhock': 2, 'marigold': 3 } #class 분류

```

train, test set을 아까 저장한 이미지 list에서 하나씩 load해서 array로 바꾸고, VGG 모델을 사용하기 위한 전처리 함수인 preprocess\_input을 이용해서 전처리 시켰다.

그리고 해당하는 클래스를 저장하기 위해서 해당 곳 폴더명을 불러와 class를 분류했다.

```
#train set
for path in train_img:
    #print(path)
    img = image.load_img(path, target_size=(img_size, img_size))
    img = image.img_to_array(img)

    x_train.append(preprocess_input(img)) #VGG 모델을 위한 전처리
    #class 분류 숫자 저장
    sp = str(path).split(sep='\\')
    y_train.append(y.get(sp[9]))

#test set
for path in test_img:
    img = image.load_img(path, target_size=(img_size, img_size))
    img = image.img_to_array(img)
    x_test.append(preprocess_input(img)) #VGG 모델을 위한 전처리
    #class 분류 숫자 저장
    sp = str(path).split(sep='\\')
    y_test.append(y.get(sp[9]))
```

loss는 categorical\_crossentropy할 것이기에 y들은 원-핫 인코딩했으며, 잘 저장되었는지 확인했다(train은 shuffle 하기 전에 확인 후 주석 처리). 그리고 list를 np.array로 변경했다.

```
#원-핫 인코딩
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# 잘 저장 되었는지 확인
# for i in X:
#     print(i)
# for i in x_train:
#     print(i)
# for i in x_test:
#     print(i)
# for i in y_train:
#     print(i)
```



오버피팅이 일어나면 그 전 데이터를 test로 돌리기 위해서 가중치를 1 epoch마다 저장하게 했다.

그리고 opt를 설정하고 compile로 학습 환경을 설정했다. epoch은 6, 30도 해봤는데 15 정도가 6은 너무 적게 돌려지고 30은 너무 오래 걸리기도 하고, 그 전에 오버 피팅이 일어나서 15 정도가 적당한 것 같았다. batch\_size를 64로 했더니 메모리 초과가 떠서 32 이하로 진행할 것이다.

```
opt = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(optimizer = opt, loss="categorical_crossentropy",
metrics=['accuracy'])

#모델 전체 저장
if not os.path.exists("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/ckpt/zero/RES"):
    os.mkdir("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/ckpt/zero/RES")
model.save("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/ckpt/zero/RES/zero.h5")
#체크포인트, 가중치만 저장, 1 epoch마다 저장
filepath = "C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/ckpt/zero/zero-{epoch:04d}.ckpt"
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath, verbose=0,
save_best_only=True, save_weights_only=True, save_freq='epoch',
monitor='val_accuracy')

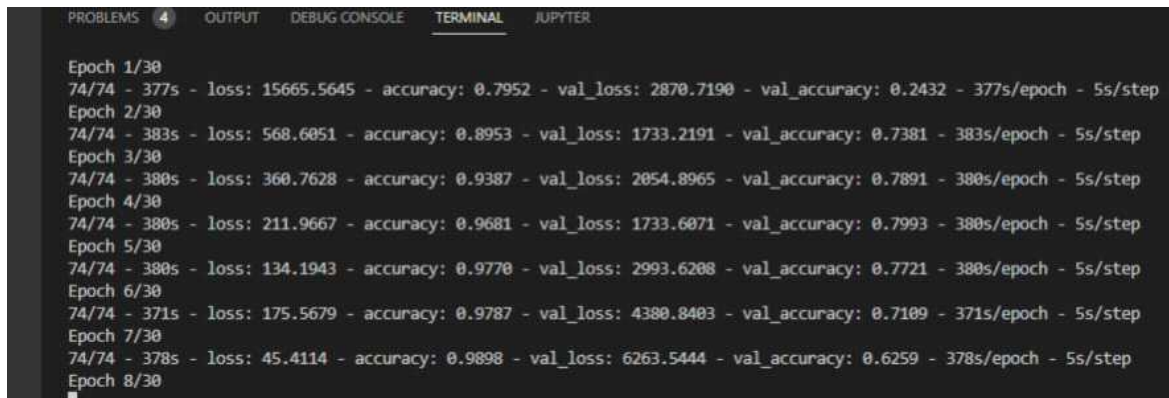
#batch_size를 64로 하니까 할당이 메모리 10%를 초과한다고 떠서 32로 변경
ret = model.fit(x_train, y_train, epochs=15, batch_size=32, validation_split=0.2,
verbose=2, callbacks=[cp_callback])
```

loss, acc를 볼 그래프 출력 코드이다. 근데 거의 fit을 할 때 verbose=2로 해둬서 오래 걸리거나 오버피팅 일 때 해당 숫자를 보면서 끊어가며 했다.

```
fig, ax = plt.subplots(1, 2, figsize=(10, 6))
ax[0].plot(ret.history['loss'], "b-", label="train loss")
ax[0].plot(ret.history['val_loss'], "r-", label="var loss")
ax[0].set_title("loss")
ax[0].set_xlabel("epochs")
ax[0].set_ylabel("loss")
```

```
ax[1].plot(ret.history['accuracy'], "b-",
          label="train accuracy")
ax[1].plot(ret.history['val_accuracy'], "r-",
          label="val accuracy")
ax[1].set_title("accuracy")
ax[1].set_xlabel("epochs")
ax[1].set_ylabel("accuracy")
plt.legend(loc="best")
fig.tight_layout()
plt.show()
```

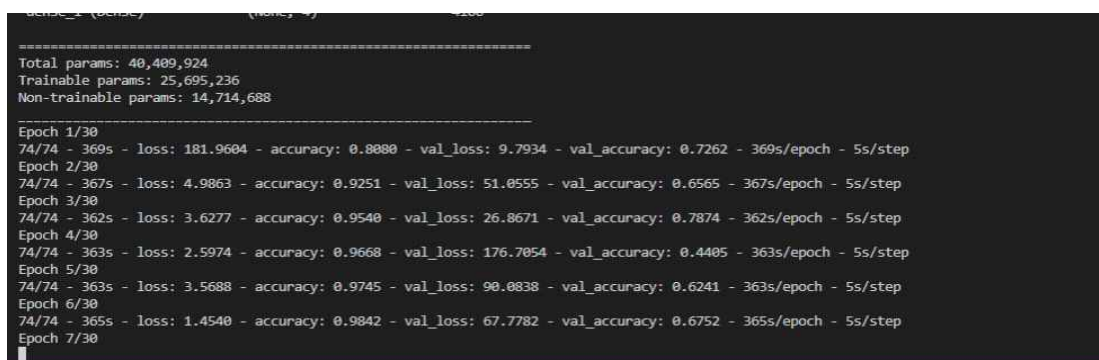
## 2.3 실험



```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
Epoch 1/30
74/74 - 377s - loss: 15665.5645 - accuracy: 0.7952 - val_loss: 2870.7190 - val_accuracy: 0.2432 - 377s/epoch - 5s/step
Epoch 2/30
74/74 - 383s - loss: 568.6051 - accuracy: 0.8953 - val_loss: 1733.2191 - val_accuracy: 0.7381 - 383s/epoch - 5s/step
Epoch 3/30
74/74 - 380s - loss: 360.7628 - accuracy: 0.9387 - val_loss: 2054.8965 - val_accuracy: 0.7891 - 380s/epoch - 5s/step
Epoch 4/30
74/74 - 380s - loss: 211.9667 - accuracy: 0.9681 - val_loss: 1733.6071 - val_accuracy: 0.7993 - 380s/epoch - 5s/step
Epoch 5/30
74/74 - 380s - loss: 134.1943 - accuracy: 0.9770 - val_loss: 2993.6208 - val_accuracy: 0.7721 - 380s/epoch - 5s/step
Epoch 6/30
74/74 - 371s - loss: 175.5679 - accuracy: 0.9787 - val_loss: 4380.8403 - val_accuracy: 0.7109 - 371s/epoch - 5s/step
Epoch 7/30
74/74 - 378s - loss: 45.4114 - accuracy: 0.9898 - val_loss: 6263.5444 - val_accuracy: 0.6259 - 378s/epoch - 5s/step
Epoch 8/30
```

-> 위의 코드들을 돌려보니 loss가 처음부터 크게 잡히고, epoch 5부터 validation loss가 오르고 acc가 내려가기 시작했다.

이 부분부터 해결해야겠다 싶어서 크게 잡은 learning\_rate를 0.1에서 0.01로 변경했다.



```
dense_1 (dense) (None)
Total params: 40,409,924
Trainable params: 25,695,236
Non-trainable params: 14,714,688
Epoch 1/30
74/74 - 369s - loss: 181.9604 - accuracy: 0.8080 - val_loss: 9.7934 - val_accuracy: 0.7262 - 369s/epoch - 5s/step
Epoch 2/30
74/74 - 367s - loss: 4.9863 - accuracy: 0.9251 - val_loss: 51.0555 - val_accuracy: 0.6565 - 367s/epoch - 5s/step
Epoch 3/30
74/74 - 362s - loss: 3.6277 - accuracy: 0.9540 - val_loss: 26.8671 - val_accuracy: 0.7874 - 362s/epoch - 5s/step
Epoch 4/30
74/74 - 363s - loss: 2.5974 - accuracy: 0.9668 - val_loss: 176.7054 - val_accuracy: 0.4405 - 363s/epoch - 5s/step
Epoch 5/30
74/74 - 363s - loss: 3.5688 - accuracy: 0.9745 - val_loss: 90.0838 - val_accuracy: 0.6241 - 363s/epoch - 5s/step
Epoch 6/30
74/74 - 365s - loss: 1.4540 - accuracy: 0.9842 - val_loss: 67.7782 - val_accuracy: 0.6752 - 365s/epoch - 5s/step
Epoch 7/30
```

-> loss가 아까보다 많이 줄었지만 val\_loss가 안정적이지가 않다.

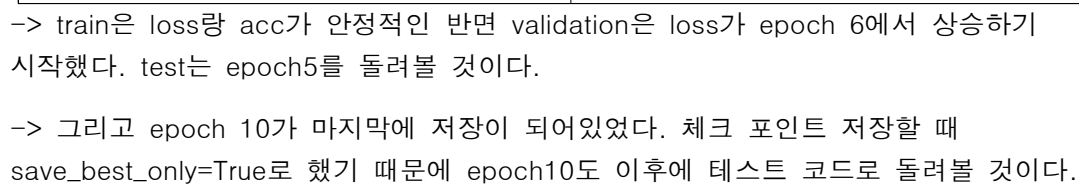
(이후에 test 코드에서 돌릴 case 0)

optimizer를 Adam으로 바꿔서 해보았다. 추가적으로 ouput 층에서 Dense층을 원래 VGG



```
# 이전
# x = Dense(1024, activation = 'relu')(x)
# outs = Dense(4, activation = 'softmax')(x)

# 변경된 것
x = Dense(4096, activation = 'relu')(x)
x = Dense(4096, activation = 'relu')(x)
outs = Dense(4, activation = 'softmax')(x)
```

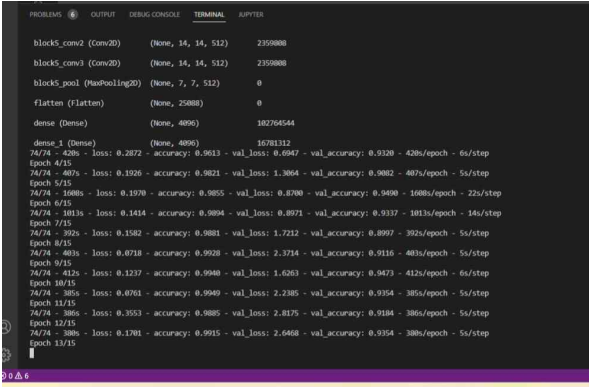


```

INFO: abt1-ctrl-vgd@1000: 1000%
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
block5_conv2 (Conv2D) (None, 14, 14, 512) 2359888
block5_conv3 (Conv2D) (None, 14, 14, 512) 2359888
block5_pool (MaxPooling2D) (None, 7, 7, 512) 0
Flatten (Flatten) (None, 25088) 0
dense (Dense) (None, 4096) 182764544
dense_1 (Dense) (None, 4096) 16781312
dense_2 (Dense) (None, 4) 16388
=====
Total params: 134,276,932
Trainable params: 119,562,244
Non-trainable params: 14,714,688
=====
Epoch 1/15
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
74/74 - 420s - loss: 19300.9941 - categorical_accuracy: 0.6377 - val_loss: 1.4335 - val_categorical_accuracy: 0.7398 - 420s/epoch - 6s/step
Epoch 2/15

```

metrics는 기존의 accuracy로 두고, learning\_rate를 0.001로 한 뒤 optimizer는 Adam, RMSprop을 비교했다.

Adam	RMSprop
<p>결과 사진이 날라갔지만 epoch 4 이후부터는 썩 좋은 결과는 아니었다.</p>	 <p>-&gt; epoch 5 이후부터 좋지 않았고 val_loss도 zero 테스트 상태보다 내려가지 못했다.</p>

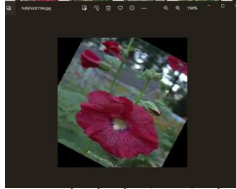
case 0 테스트 상태에서 Dense층을 하나 삭제해서 해보았다.

<pre># 이전 # x = Dense(4096, activation = 'relu')(x) # x = Dense(4096, activation = 'relu')(x) # outs = Dense(4, activation = 'softmax')(x)  # 변경된 것 x = Dense(4096, activation = 'relu')(x) outs = Dense(4, activation = 'softmax')(x)</pre>
--

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	JUPYTER
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0		
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359888		
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359888		
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359888		
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0		
flatten (Flatten)	(None, 25088)	0		
dense (Dense)	(None, 4096)	102764544		
dense_1 (Dense)	(None, 4)	16388		
<pre>total params: 117,495,620 trainable params: 102,780,932 non-trainable params: 14,714,688</pre>				
<pre>epoch 1/15 4/74 - 391s - loss: 404.8285 - accuracy: 0.7756 - val_loss: 55.3693 - val_accuracy: 0.7313 - 391s/epoch - 5s/step epoch 2/15 4/74 - 797s - loss: 15.0032 - accuracy: 0.9089 - val_loss: 9.6846 - val_accuracy: 0.8912 - 797s/epoch - 11s/step epoch 3/15 4/74 - 1315s - loss: 5.9627 - accuracy: 0.9566 - val_loss: 19.9319 - val_accuracy: 0.8861 - 1315s/epoch - 18s/step epoch 4/15 4/74 - 1338s - loss: 4.3500 - accuracy: 0.9642 - val_loss: 16.1438 - val_accuracy: 0.9252 - 1338s/epoch - 18s/step epoch 5/15 4/74 - 1348s - loss: 3.3884 - accuracy: 0.9888 - val_loss: 114.5152 - val_accuracy: 0.8231 - 1348s/epoch - 18s/step epoch 6/15</pre>				

-> 안 하는 게 좋을 듯하다.

case 0 상태에서 data\_augmentation rotate를 사용해서 학습 데이터 수를 대폭 늘리고 학습시켰다.



-> 회전시킨 예시 사진이다. 한 사진당 2번씩 랜덤 각도로 회전시켰다. 각 class 별로 2000~3000장 정도가 이번에 사용될 train 이미지이다.

rotate 코드는 아래와 같다.

```
import random
import os
from PIL import Image

file_path      =      r'C:\Users\freet\Desktop\3학년2학기\인공지능\과제\꽃      분류
\train_data_rotate\marigold' # 늘릴 해당 파일 경로
file_names = os.listdir(file_path) # 파일에 저장된 이름 저장
total_origin_image_num = len(file_names) # 파일 수
augment_cnt = 1 # 이름 뒤에 붙일 cnt

for file_name in file_names: # 파일들 하나씩 불러오기
    if file_name.endswith('.jpg'): # 확장자가 jpg일때만

        origin_image_path = file_path + '/' + file_name # 원래 경로에 이미지 이
름 합친 경로
        save_image_path  =  r'C:\Users\freet\Desktop\3학년2학기\인공지능\과제\꽃
분류\train_data_rotate\marigold' # 저장할 이미지

        image = Image.open(origin_image_path) # 이미지 불러오기
        # 이미지 기울이기
        rotated_image = image.rotate(
            random.randrange(-30, 0)) # 각도 -30 ~ 0에서 난수 추출
        rotated_image.save(save_image_path + 'marigold' +
                           str(augment_cnt) + '_rotated1.jpg') # 이미지 저장

        rotated_image2 = image.rotate(
            random.randrange(0, 30)) # 각도 0 ~ -30에서 난수 추출
        rotated_image2.save(save_image_path + 'marigold' +
```

```
str(augment_cnt) + '_rotated2.jpg') # 이미지 저장
augment_cnt += 1 # 이름 뒤에 붙일 cnt 갯수 늘리기
```

# imgaug 라이브러리를 사용해서 이미지를 늘리기도 해서 나중에 해보기

늘린 이미지들로 돌려보니까 결과가 val\_loss보다 train\_loss가 더 높게 나왔다. 찾아보니

**첫 번째 이유:** 훈련 중에는 Regularization이 적용되었으나, 검증/테스트 중에는 적용되지 않았다.

우리는 모델을 학습시킬 때 정규화를 활용한다. 정규화를 활용하면 다음과 같은 효과를 얻을 수 있다.

- 더 높은 검증/테스트 정확도를 확보할 수 있다.
- 검증 및 테스트 세트 외부 데이터로 더 잘 일반화할 수 있다.

정규화 방법은 종종 검증/테스트 정확도 개선을 위해 훈련 정확도를 희생한다. 어떤 경우엔 검증 손실이 훈련 손실보다 낮아질 수 있다. 그리고, dropout과 같은 정규화 방법은 검증/테스트 시 적용되지 않는다는 점에 유의하라.

\* \* \*

**두 번째 이유:** 각 Epoch 동안 Training Loss를 측정하고, 각 Epoch 후에 Validation Loss를 측정한다.

훈련 손실은 전체 epoch동안 지속적으로 보고되나, 검증 metric은 현재 훈련 에포크가 완료된 후에만 검증 세트에 대해 계산된다.

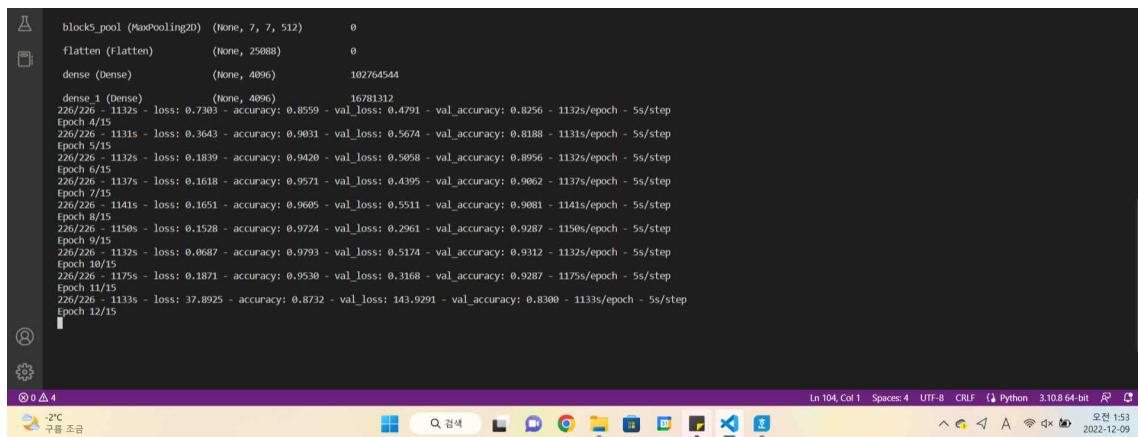
\* \* \*

**세 번째 이유:** 검증 세트가 훈련 세트보다 쉬울 수 있거나 누출이 있을 수 있다.

- 검증 세트가 훈련 세트와 동일한 분포에서 샘플링 되었음을 보장할 수 있는가?
- 검증 예제가 훈련 이미지만큼 어렵다 확인하는가?
- 훈련 샘플이 실수로 검증/테스트 샘플과 혼합되지 않았다 확인할 수 있는가?
- 코드가 훈련, 검증 및 테스트 분할을 올바르게 생성했다 확인하는가?

라는데 세 번째 이유일 것 같다. 클래스 4개를 무작위로 20% validation set으로 쓰고있으니 말이다. train set에서 각 클래스 평균 개수인 2182장의 20%인 400장 정도를 각 클래스마다 뽑아와서 validation set 폴더로 따로 뒀다. (최종 코드는 부록)

Adam이 아닌 RMSprop으로 실수로 잘못 돌렸는데, 과적합이 epoch 6부터 발생했었다. 아래의 사진은 원래 하려던 Adam으로 돌린 것이다.



-> 아까들과 다르게 loss들이 꾸준히 하락하는 모습을 보여준다.

-> epoch11에서 loss들이 급격하게 상승해서 중간에 멈췄다. epoch9부터 val\_loss가 슬금슬금 오르더니 epoch 10부터 train\_loss도 올랐다. 그래서 test 코드에서 epoch8, epoch 9를 돌려보려 한다. (이후에 test 코드에서 돌릴 case 1)

loss를 mse로 변경했는데 더 안 좋게 나왔어서 case 1에서 lr을 0.001로 변경해서 해봤다.

```
88 model.summary()
89
90
91 opt = tf.keras.optimizers.Adam(learning_rate=0.001) #0.001
92 model.compile(optimizer = opt, loss="categorical_crossentropy", metrics=['accuracy']) # categorical_crossentropy, mse
93
94 #모델 전체 저장
95 if not os.path.exists("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류/ckpt/two/RES"):
    block5_conv2 (Conv2D) (None, 14, 14, 512) 2359808
    block5_conv3 (Conv2D) (None, 14, 14, 512) 2359808
    block5_pool (MaxPooling2D) (None, 7, 7, 512) 0
    flatten (Flatten) (None, 25088) 0
    226/226 - loss: 0.0663 - accuracy: 0.9828 - val_loss: 0.3849 - val_accuracy: 0.9362 - 1169s/epoch - 5s/step
    Epoch 4/15
    226/226 - loss: 0.0904 - accuracy: 0.9825 - val_loss: 0.4304 - val_accuracy: 0.9480 - 1172s/epoch - 5s/step
    Epoch 5/15
    226/226 - loss: 0.1090 - accuracy: 0.9865 - val_loss: 0.4350 - val_accuracy: 0.9100 - 1173s/epoch - 5s/step
    Epoch 6/15
    226/226 - loss: 0.0377 - accuracy: 0.9918 - val_loss: 0.3543 - val_accuracy: 0.9475 - 1198s/epoch - 5s/step
    Epoch 7/15
    226/226 - loss: 0.0317 - accuracy: 0.9931 - val_loss: 0.3008 - val_accuracy: 0.9531 - 1192s/epoch - 5s/step
    Epoch 8/15
    226/226 - loss: 0.0085 - accuracy: 0.9982 - val_loss: 0.3377 - val_accuracy: 0.9513 - 1160s/epoch - 5s/step
    Epoch 9/15
    226/226 - loss: 0.0315 - accuracy: 0.9942 - val_loss: 0.3863 - val_accuracy: 0.9544 - 1170s/epoch - 5s/step
    Epoch 10/15
```

-> case1보다 acc가 더 높게 올라가는 모습을 볼 수 있다. loss가 작기도 하고 acc가 높은 epoch 7과 epoch 9를 테스트로 돌려볼 것이다.

(이후에 test 코드에서 돌릴 case 2)

### 3. 실험 결과

#### 3.1 test (test 코드 설명) 2.2에서의 로드 및 전처리는 생략

-> test set(class당 3장 총 12장)

train 코드에서 저장한 모델을 전체 로드하고, 해당 weights를 평가 예측했다.

그리고 클래스 별로 얼마나 예측했는지 확인했다.

```
import pathlib
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input, VGG16

#사이즈 지정
img_size = 224
```

```

test_path = pathlib.Path('C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/test_data')
test_img = list(test_path.glob('*/*.jpg'))

x_test, y_test, y_class = [], [], []
y = {'cosmos': 0, 'daisy': 1, 'hollyhock': 2, 'marigold':3 } #class 분류

#test set
for path in test_img:
    img = image.load_img(path, target_size=(img_size, img_size))
    img = image.img_to_array(img)
    x_test.append(preprocess_input(img)) #VGG 모델을 위한 전처리
    #class 분류 숫자 저장
    sp = str(path).split(sep='\\')
    y_class.append(sp[9]) #이후에 어떤 클래스가 잘 예측했는지에 사용
    y_test.append(y.get(sp[9]))

y_test = tf.keras.utils.to_categorical(y_test)

x_test = np.array(x_test)
y_test = np.array(y_test)

# for i in x_test:
#     print(i)
# for i in y_test:
#     print(i)

#모델 전체 로드
model = tf.keras.models.load_model("C:/Users/freet/Desktop/3학년2학기/인공지능/
과제/꽃 분류/ckpt/two/RES/two.h5")

#weights
#latest = tf.train.load_checkpoint("C:/Users/freet/Desktop/3학년2학기/인공지능/과
제/꽃 분류/ckpt/zero/")
model.load_weights(r"C:\Users\freet\Desktop\3학년2학기\인공지능\과제\꽃 분류
\ckpt\two\two-0007.ckpt")

```

```

#모델 평가 예측
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

y_pred = model.predict(x_test, verbose=0)
y_pred = np.around(y_pred)
y_act = np.around(y_test)

pred_class = {'cosmos': 0, 'daisy': 0, 'hollyhock': 0, 'marigold':0 } #class 분류
for i in range(len(y_pred)):
    print(y_class[i], ": ", end='')
    # print(y_pred[i])
    # print(y_act[i])
    if(np.all(y_pred[i] == y_act[i])):
        print("true")
    else:
        print("false")

```

(case 0 테스트)

ALL

```

1/1 - 2s - loss: 0.2561 - accuracy: 0.8333 - 2s/epoch - 2s/step
cosmos : true
cosmos : true
cosmos : true
daisy : true
daisy : true
daisy : true
hollyhock : true
hollyhock : true
hollyhock : false
marigold : false
marigold : true
marigold : true

```

-> epoch 5일 때 test\_loss는 0.25, test\_acc는 0.83이다.

```

To enable them in other operations, rebuild
1/1 - 1s - loss: 1.8909 - accuracy: 0.8333 -
cosmos : true
cosmos : true
cosmos : true
daisy : true
daisy : true
daisy : true
hollyhock : true
hollyhock : true
hollyhock : false
marigold : false
marigold : true
marigold : true

```

-> epoch 10일 때 test\_loss는 1.89, test\_acc는 0.83이다.

(case 1 테스트)

ALL									
<pre>ns in performance-critical operations: AVX AVX2 To enable them in other operations, rebuild TensorFlow with the app 1/1 - 2s - loss: 1.3503 - accuracy: 0.9167 - 2s/epoch - 2s/step PS C:\Users\freet&gt; &amp; C:/Users/freet/AppData/Local/Programs/Python/P 2022-12-09 18:51:58.283211: I tensorflow/core/platform/cpu_feature_ ns in performance-critical operations: AVX AVX2 To enable them in other operations, rebuild TensorFlow with the app 1/1 - 2s - loss: 1.2672 - accuracy: 0.9167 - 2s/epoch - 2s/step</pre> <p>-&gt; epoch 8일 때 test_loss는 1.35 test_acc는 0.91</p> <p>-&gt; epoch 9일 때 test_loss는 1.26 test_acc는 0.91</p> <p>--&gt; 이전의 case 0에서 epoch 5, 10 사이의 loss 값들이 나오지만 acc가 상승한 모습을 보여준다.</p> <p>epoch 9일 때(각 class 당 맞춘 개수)</p> <pre>1/1 - 2s - loss: 1.3503 - accuracy: 0.9167 - 2s/epoch - 2s/step cosmos : true cosmos : true cosmos : true daisy : true daisy : true daisy : true hollyhock : true hollyhock : true hollyhock : false marigold : true marigold : true marigold : true</pre> <p>-&gt; 클래스 당 3장 총 12장</p> <table><tr><td>코스모스</td><td>데이지</td></tr><tr><td>3/3</td><td>3/3</td></tr><tr><td>접시꽃</td><td>메리골드</td></tr><tr><td>2/3</td><td>3/3</td></tr></table>		코스모스	데이지	3/3	3/3	접시꽃	메리골드	2/3	3/3
코스모스	데이지								
3/3	3/3								
접시꽃	메리골드								
2/3	3/3								

(case 2 테스트)

ALL	
<pre>PS C:\Users\freet&gt; &amp; C:/Users/freet/AppData/Local/Programs/Python/Python310/python 2022-12-09 18:56:05.650581: I tensorflow/core/platform/cpu_feature_guard.cc:193] T ns in performance-critical operations: AVX AVX2 To enable them in other operations, rebuild TensorFlow with the appropriate compi 1/1 - 2s - loss: 0.1024 - accuracy: 1.0000 - 2s/epoch - 2s/step PS C:\Users\freet&gt; &amp; C:/Users/freet/AppData/Local/Programs/Python/Python310/python 2022-12-09 18:56:16.803593: I tensorflow/core/platform/cpu_feature_guard.cc:193] T ns in performance-critical operations: AVX AVX2 To enable them in other operations, rebuild TensorFlow with the appropriate compi 1/1 - 1s - loss: 1.5151 - accuracy: 0.9167 - 1s/epoch - 1s/step</pre> <p>-&gt; epoch 7일 때 test_loss는 0.10, test_acc는 1.00</p> <p>-&gt; epoch 9일 때 test_loss는 1.51, test_acc는 0.91</p> <p>--&gt; epoch 7은 test_loss도 0.10으로 굉장히 작으며 acc가 1이다..!</p> <p>--&gt; epoch 9는 이전의 case 1과 같은 정확도를 가지며 loss가 높은 것을 볼 수 있다</p> <p>epoch 7일 때(각 class 당 맞춘 확률)</p>	



	<pre> 1/1 - 1s - loss: 0.1024 - accuracy: 1.0000 - 1s/ cosmos : true cosmos : true cosmos : true daisy : true daisy : true daisy : true hollyhock : true hollyhock : true hollyhock : true marigold : true marigold : true marigold : true </pre>	
-> test set은 한 클래스 당 3장, 총 12장		
코스모스, 데이지, 접시꽃, 메리골드		
3/3		

#### 4. 결론(결론, 과제 수행 느낌 등)

##### <결론>

데이터 수를 늘리기 전인 case0은 학습되지 않은 test set을 평균 80%로 맞출 수 있었다. 하지만 case0의 경우에 train, val의 loss가 들쭉날쭉하고 안정적이지 못했다.

데이터 수를 늘린 case1 epoch9의 경우에는 case0과 비교해서 train, val의 loss가 안정적이게 되었고, test\_acc가 늘었다. 그러나 test\_loss는 case0보다 저조했다.

case1에서 lr을 0.001로 변경했던 case2 epoch7의 경우에는 train, val, test의 acc, loss가 case0, case1보다 좋았고 test set을 모두 맞췄다.

##### <과제 수행 느낌>

한 번 결과 보는 거에 시간이 너무 걸려서 include\_top(classfication)을 많이 바꿔보고 싶었는데 못했던 게 아쉬웠다.

#### 참고문헌

- keras document callbacks:

<https://keras.io/ko/callbacks/>

- tensorflow document Checkpoint:

[https://www.tensorflow.org/api\\_docs/python/tf/train/Checkpoint](https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint)

- 김동근, 『텐서플로 딥러닝 프로그래밍』, 가메출판사, setp14(모델 저장 및 로드), setp18(영상 로드, 저장, 변환), chapter 11(사전학습 모델)

(폴더별로 파일명 맞추기, 사진 rotate하기, 이미지 사이즈 맞추기 코드는 여름방학 때 만들어 놓은 거라 어디 사이트 코드를 참고했는지 모르겠어요)

## 부록(소스코드) <최종 코드>

### train\_file.py

```
#45_2, 45_3, 45_04, 49_05, 50_01 참고.
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pathlib
import random
import os

from tensorflow.keras.layers import Input, Dense, Flatten,
GlobalAveragePooling2D, Dropout
from tensorflow.keras.applications.vgg16 import preprocess_input, VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image

#This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations: AVX AVX2 To enable them in other operations, rebuild TensorFlow
with the appropriate compiler flags.
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

#사이즈 지정
img_size = 224

#train, test path
#C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류/train_data
#C:\Users\freet\Desktop\3학년2학기\인공지능\과제\꽃 분류\train_data_rotate
train_path = pathlib.Path(r'C:\Users\freet\Desktop\3학년2학기\인공지능\과제\꽃 분
류\train_data_rotate')
val_path = pathlib.Path(r'C:\Users\freet\Desktop\3학년2학기\인공지능\과제\꽃 분
류\val_data_rotate')

#이미지 list에 저장
train_img = list(train_path.glob('*/*.jpg'))
val_img = list(val_path.glob('*/*.jpg'))
#validation set이 생겨서 shuffle 불필요
# random.shuffle(train_img)

#영상 읽어오기
```

```

x_train, y_train = [], []
x_val, y_val = [], []
y = {'cosmos': 0, 'daisy': 1, 'hollyhock': 2, 'marigold':3 } #class 분류
#train set
for path in train_img:
    #print(path)
    img = image.load_img(path, target_size=(img_size, img_size))
    img = image.img_to_array(img)

    x_train.append(preprocess_input(img)) #VGG 모델을 위한 전처리
    #class 분류 숫자 저장
    sp = str(path).split(sep='\\')
    y_train.append(y.get(sp[9]))
#val set
for path in val_img:
    #print(path)
    img = image.load_img(path, target_size=(img_size, img_size))
    img = image.img_to_array(img)

    x_val.append(preprocess_input(img)) #VGG 모델을 위한 전처리
    #class 분류 숫자 저장
    sp = str(path).split(sep='\\')
    y_val.append(y.get(sp[9]))

#원-핫 인코딩
y_train = tf.keras.utils.to_categorical(y_train)
y_val = tf.keras.utils.to_categorical(y_val)
# 잘 저장 되었는지 확인
# for i in X:
#     print(i)
# for i in x_train:
#     print(i)
# for i in y_train:
#     print(i)

#list를 np.array로 변경
x_train = np.array(x_train)
y_train = np.array(y_train)
x_val = np.array(x_val)
y_val = np.array(y_val)

```

```

#input
inputs = Input(shape=(img_size, img_size, 3))
#model 불러오기
model = VGG16(weights="imagenet", classes=4, input_shape=(img_size, img_size, 3), include_top=False, input_tensor=inputs)
model.trainable = False

#output
x = model.output
x = Flatten()(x)
# x = Dense(1024, activation = 'relu')(x)
# outs = Dense(4, activation = 'softmax')(x)
x = Dense(4096, activation = 'relu')(x)
x = Dense(4096, activation = 'relu')(x)
outs = Dense(4, activation = 'softmax')(x)
model = tf.keras.Model(inputs, outs)
model.summary()

opt = tf.keras.optimizers.Adam(learning_rate=0.001) #0.001
model.compile(optimizer = opt, loss="categorical_crossentropy",
metrics=['accuracy']) # categorical_crossentropy, mse

#모델 전체 저장
if not os.path.exists("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/ckpt/two/RES"):
    os.mkdir("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/ckpt/two/RES")
model.save("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/ckpt/two/RES/two.h5")
#체크포인트, 가중치만 저장, 1 epoch마다 저장
filepath = "C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/ckpt/two/two-{epoch:04d}.ckpt"
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath, verbose=0,
save_best_only=True, save_weights_only=True, save_freq='epoch',
monitor='val_accuracy')

#batch_size를 64로 하니까 할당이 메모리 10%를 초과한다고 떼서 32로 변경
ret = model.fit(x_train, y_train, epochs=15, batch_size=32, validation_data=(x_val,

```

```

y_val), verbose=2, callbacks=[cp_callback])

fig, ax = plt.subplots(1, 2, figsize=(10, 6))
ax[0].plot(ret.history['loss'], "b-", label="train loss")
ax[0].plot(ret.history['val_loss'], "r-", label="var loss")
ax[0].set_title("loss")
ax[0].set_xlabel("epochs")
ax[0].set_ylabel("loss")

ax[1].plot(ret.history['accuracy'], "b-",
            label="train accuracy")
ax[1].plot(ret.history['val_accuracy'], "r-",
            label="val accuracy")
ax[1].set_title("accuracy")
ax[1].set_xlabel("epochs")
ax[1].set_ylabel("accuracy")
plt.legend(loc="best")
fig.tight_layout()
plt.show()

```

### test\_file.py

```

import pathlib
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input, VGG16

#사이즈 지정
img_size = 224

test_path = pathlib.Path('C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류
/test_data')
test_img = list(test_path.glob('*/*.jpg'))

x_test, y_test, y_class = [], [], []
y = {'cosmos': 0, 'daisy': 1, 'hollyhock': 2, 'marigold':3 } #class 분류

```

```

#test set
for path in test_img:
    img = image.load_img(path, target_size=(img_size, img_size))
    img = image.img_to_array(img)
    x_test.append(preprocess_input(img)) #VGG 모델을 위한 전처리
    #class 분류 숫자 저장
    sp = str(path).split(sep='\\')
    y_class.append(sp[9]) #이후에 어떤 클래스가 잘 예측했는지에 사용
    y_test.append(y.get(sp[9]))

y_test = tf.keras.utils.to_categorical(y_test)

x_test = np.array(x_test)
y_test = np.array(y_test)

# for i in x_test:
#     print(i)
# for i in y_test:
#     print(i)

#모델 전체 로드
model = tf.keras.models.load_model("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류/ckpt/two/RES/two.h5")

#weights
#latest = tf.train.load_checkpoint("C:/Users/freet/Desktop/3학년2학기/인공지능/과제/꽃 분류/ckpt/zero/")
model.load_weights(r"C:\Users\freet\Desktop\3학년2학기\인공지능\과제\꽃 분류\ckpt\two\two-0007.ckpt")

#모델 평가 예측
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

y_pred = model.predict(x_test, verbose=0)
y_pred = np.around(y_pred)
y_act = np.around(y_test)

```

```
pred_class = {'cosmos': 0, 'daisy': 0, 'hollyhock': 0, 'marigold':0 } #class 분류
for i in range(len(y_pred)):
    print(y_class[i], ": ", end='')
    # print(y_pred[i])
    # print(y_act[i])
    if(np.all(y_pred[i] == y_act[i])):
        print("true")
    else:
        print("false")
```