통계학 특강 Final Report

고려대학교 정경대학 통계학과

2014150028 조영상

I. 개요

1) 분석배경

한 학기동안 통계학 특강을 수강하며 가장 인상깊었던 딥러닝의 효과는 이미지 데이터 분석 시 상당히 높은 퍼포먼스를 보여준 것이다. 실제로 이미지 데이터 분석에 사용될 모형은 하나의 계수의 효과를 해석하는 것보다 분류 정확도와 같은 정확도, 예측력의 최대화하는 모형이 적합하다. 따라서 이미지 데이터 분석을 하는 다른 머신러닝 방법인 주성분 분석(PCA), 비음수 행렬 분해(NMF)등 보다 높은 정확도를 갖는 모형을 만들어 낼 수 있는 딥러닝이 이미지 데이터 분석에 있어 가장 적합한 모형이라고 할 수 있다.

더불어, 딥러닝에서 사용하는 신경망(Neural Network)은 사람의 뇌 구조를 차용해 만든 알고리즘이다. 이는 우리가 분류하는 방식이나 인식하는 방식과 유사하게 데이터를 분석한다는 뜻이라고 할 수 있다. 때문에, 색채만 숫자로 표현되어 있는 데이터에서 사람들(흔히 미술전문가)이 일정한 패턴으로 분류해놓은 그림 데이터에 대한 분류 작업에 딥러닝을 적용할 때, 색채뿐만 아니라 색채가 가진 의미, 잠재적인 패턴등을 인식할 수 있는지 궁금하였다.

이와 같은 이유들로 이미지 데이터에 적합한 딥러닝의 방법들을 사용하여 평소에 관심이 많았던 미술작품에 대한 이미지 분석을 시도하였다.

2) 분석방향

첫째로 가장 관심이 많은 사조인 인상주의에 대한 분석을 시도하였다. 인상주의는 19c 유럽에서 시작된 미술 사조이며, RGB 데이터를 분석하는 방법을 통해 각 예술가들의 특징에 따라 예술가를 분류하는 딥러닝 모형이 분석에 적합하다고 생각한다. 뿐만 아니라 화가의 데이터 따라서 인상주의 대표 화가 10명에 대한 데이터를 화가에 따라 분류하는 모형을 만들어 보고자 하였다.

둘째로 인상주의는 색채를 강조한 미술이다. 따라서 딥러닝의 RGB데이터 분석을 통해 분류하는 법을 사용하면 미술 작품에 담긴 미술 사조간 색채의 특징을 통해 딥러닝 모형을 통해 알 수 있을지 궁금하였다. 따라서 미술 작품의 따라서 작가뿐만 아니라 미술 사조의 분석을 통해 각 예술품이 속하는 미술 사조에 따라 데이터를 분류하였다.

3) 데이터 설명

데이터는 Kaggle에서 사용한 두개의 데이터를 합쳐서 사용하였다. 먼저 Impression Data는 인상주의 화가 10명에 대한 데이터로 각 화가당 Train 데이터와 Validation 데이터가 폴더에 따라 나눠져 있었다. 두번째 데이터는 각 예술 사조에 대하여 분류하기 위해 Kaggle에 있는 데이터를 사용하였다. 3개의 사조가 각각 폴더로 나뉘어 있었으며, 이 사조에 첫번째 인상주의 데이터를 추가해 총 4개의 사조에 대한 분류를 진행하였다. 다만 화가에 따라 중복된 미술 사조가 있었다. 예를 들어 세잔의 경우 인상주의 후기와 입체주의 초기의 대표적인 미술가이다. 따라서 각 미술가들에 대한 정보를 바탕으로 세잔과 마티스를 제외한 작품들에서 인상주의 작품의 표본을 선택하여 합치도록 하였다.

	데이터	범주
1	Impression	세잔, 드가, 고갱, 하삼, 마티스, 모네, 피사로, 르누아르, 사강, 반 고흐
2	Art	입체주의, 표현주의, 낭만주의

표1. 데이터 설명

II. 본론

분석 1: 10명의 인상주의 화가에 따른 작품을 분류해보자.

(1) 데이터 전처리

분석을 시작하기에 앞서 이미지 데이터에 대한 전처리 작업을 시작하였다. 먼저 Impression 데이터의 경우 알아보기 쉽도록 그림 파일의 이름을 바꿔주는 작업을 하였다. 이때, 각 폴더의 이름에 따라 인덱스를 추가하는 함수를 만들어 사용하였다.

```
import os
def changeName(path, cName):
    i = 1
    for filename in os.listdir(path):
        print(path+filename, '=>', path+str(cName)+'_'+str(i)+'.jpg')
        os.rename(path+filename, path+str(cName)+'_'+str(i)+'.jpg')
        i += 1
```

파일의 이름을 '파일이 속한 폴더'_'숫자'.jpg 형태로 바꿔주었다. 또한 각 화가별로 그림의 개수가 일정한지 알아보았다. Train 폴더에는 두 화가(드가 398, 피사로 398)의 그림을 제외한 다른 화가 그림의 개수는 399개가 포함되어 있었고 Validation 폴더에는 일정하게 모든 화가의 그림이 99개씩 포함되어 있었다. 따라서 화가 별 그림의 개수를 맞춰줘야 하는지 알아보았다. 실제 모델 적합 결과 데이터 분석을 위해 모든 폴더의 데이터 수가 일정한지 유무에 따라 모델의 예측 정확도 차이는 없었다.

데이터 분석을 위해 이 그림들을 모두 0과 1사이 값으로 나눠주는 작업을 실시하였다. 이후 ImageDataGenerator를 통해 데이터의 수는 유지한 채, 폴더에 따라 10개의 범주로 나눠 파이썬으로 불러오는 작업을 진행하였다. 이때 불러올 그림을 모두 픽셀 크기를 150, 150으로 맞춰

주었다. 또한 범주가 10개이므로 class_mode=categorical을 사용하였다.

```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
train_datagen=ImageDataGenerator(rescale=1./255)
validation_datagen=ImageDataGenerator(rescale=1./255)
```

```
Using TensorFlow backend.
```

```
train_generator=train_datagen.flow_from_directory(directory=impressionist_train_dir,
                                                  target_size=(150,150),
                                                  batch_size=25,
                                                  class_mode='categorical')
validation_generator=validation_datagen.flow_from_directory(directory=impressionist_validation_dir,
                                                  target_size=(150,150),
                                                  batch_size=10,
                                                  class_mode='categorical')
```

```
Found 3988 images belonging to 10 classes.
Found 990 images belonging to 10 classes.
```

(2) 모델 적용

1. 기본 모델

이후 가장 간단한 분석을 위해 CNN층과 마지막 MLP층으로만 구성된 모델에 적합해 보았다. 이때 예측하려고 하는 범주가 10종류 이므로 마지막 MLP 층의 노드 수는 10이며 이때의 활성함수는 softmax를 사용하였다. 따라서 모델 적합에서 categorical_crossentropy를 손실함수로 사용하였다. 또한 maxpooling을 사용하여 픽셀의 수를 반으로 줄여 모수를 조절해 주었다.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 15, 15, 256)       295168
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 7, 7, 256)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_11 (Dense)             (None, 512)               6423040
_________________________________________________________________
dense_12 (Dense)             (None, 10)                5130
=================================================================
Total params: 6,816,586
Trainable params: 6,816,586
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 15, 15, 256)       295168
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 7, 7, 256)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 7, 7, 256)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_7 (Dense)              (None, 32)                401440
_________________________________________________________________
dense_8 (Dense)              (None, 10)                330
=================================================================
Total params: 790,186
Trainable params: 790,186
Non-trainable params: 0
_________________________________________________________________
```

그림1. (좌)출력층 직전 노드가 512 (우)Dropout을 적용하고 출력층 직전 노드가 32

그림1 (좌)의 모델에 적합해본 결과 상당한 과대적합이 발생하였다. (좌)모델의 경우 acc: 0.8867, val_acc: 0.3612이다. 따라서 출력층 그림1 (우) 모델과 같이 Dropout을 적용하고 직전의 노드를 32로 줄여서 모델을 만들어 보았다. 또한 모수가 줄어 정확도가 떨어질 것을 우려해 직전 (좌)의 모델에서 사용한 Epoch 수보다 큰 수의 Epoch을 적용하였다. (우) 모델의 경우 acc: 0.4132, val_acc: 0.3576으로 과대적합이 상당히 해결된 것을 알 수 있었다. 다만, 과대적합이 확실히 해결된 것이 아닌 것과 더불어 전체적인 정확도가 매우 낮은 것을 알 수 있다.

2. 데이터 증대 모델(Augment Model)

이러한 결과가 나온 이유는 적합에 사용된 데이터의 개수가 범주에 비해 매우 적은 것 때문에 발생한 문제이다. 따라서 데이터 증대를 이용하였다. 데이터를 증가시키기 위해 ImageDataGenerator를 사용하여 그림에 회전과 이동, 뒤집기를 적용하였다. 하지만 그림을 최대한 그대로 유지시키는 것이 중요하며, 이러한 특성을 반영하기 위해 그림을 이동시키며 발생한 빈 공간을 그림의 다른 부분으로 채워주는 방법을 이용하였다.

이 부분은 ImageDataGenerator의 옵션 중 fill_node를 통해 조절해줄 수 있다. fill_mode는 총 4가지로 구성되어 있다. 이 중 constant와 nearest는 빈 공간을 검은색과 그림이 이동하며 남긴 잔상으로 채운다. 나머지 reflect와 wrap은 빈 공간을 그림의 다른 부분으로 채워준다. 실제 그림에 적용한 결과는 다음과 같다.

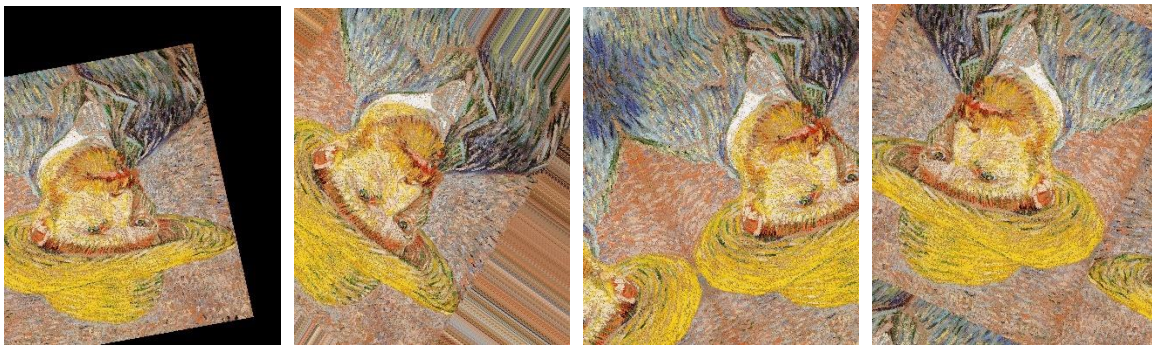

그림2. fill_node에 따른 적용결과 (1) constant (2) nearest (3) reflect (4) wrap

따라서 데이터 증대(Data Augment)를 위해 fill_mode를 reflect로 실시하였다.

```
data_aug_gen=ImageDataGenerator(rescale=1./255,
                                rotation_range=55,
                                width_shift_range=0.3,
                                height_shift_range=0.1,
                                shear_range=0.5,
                                horizontal_flip=True,
                                vertical_flip=True,
                                fill_mode='reflect')
```

데이터 증대를 위한 코드는 위와 같다. 이러한 증대를 Train 데이터에만 적용했으며, 배치 사

이즈는 16으로 설정해주었다. Validation 데이터에 포함된 그림의 형태를 바꾸면 정확한 예측이 불가능하다.

```
batch_size = 16
train_generator=data_aug_gen.flow_from_directory(impressionist_train_dir ,target_size=(150,150),
                                                 batch_size=batch_size, class_mode='categorical')
validation_generator=validation_datagen.flow_from_directory(impressionist_validation_dir ,target_size=(150,150),
                                                 batch_size=batch_size, class_mode='categorical')
```

이후 데이터 적합을 위한 모델을 사용하였다. 이때 앞과 마찬가지로 CNN 층은 4개 사용하였고, 뒤 두개의 층에 InstanceNormalization을 사용했다. 그림 데이터의 색채를 유지해주기 위해 일반 배치정규화를 사용하기보다 InstanceNormalization은 각 픽셀의 정보를 유지하고 정규화하기 때문에 그림의 정보를 잃지 않게된다. 모델은 아래 그림3과 같다.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_16 (InputLayer)        [(None, 150, 150, 3)]     0
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 15, 15, 256)       295168
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 8, 8, 256)         0
_________________________________________________________________
flatten_11 (Flatten)         (None, 16384)             0
_________________________________________________________________
dropout_11 (Dropout)         (None, 16384)             0
_________________________________________________________________
dense_21 (Dense)             (None, 32)                524320
_________________________________________________________________
dense_22 (Dense)             (None, 10)                330
=================================================================
Total params: 913,066
Trainable params: 913,066
Non-trainable params: 0
_________________________________________________________________
```

```
import instancenormalization
inputs = Input(shape=(150, 150, 3))
x = Conv2D(32, (3, 3))(inputs)
x = instancenormalization.InstanceNormalization(
    axis=-1,center=False, scale=False)(x)
x = Activation('relu')(x)
x =  MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x =  MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(128, (3, 3), activation='relu')(x)
x =  MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(256, (3, 3), activation='relu')(x)
x =  MaxPooling2D((2, 2), padding='same')(x)
x = Flatten()(x)
x = Dropout(0.6)(x)
x = Dense(32,activation='relu')(x)
x = Dense(10,activation='softmax')(x)
aug_model = Model(inputs, x)
aug_model.summary()
```

그림 3 Augment model

```
from tensorflow.keras import optimizers
aug_model.compile(loss='categorical_crossentropy',
                  optimizer=optimizers.RMSprop(lr=1e-4),
                  metrics=['acc'])
aug_result=aug_model.fit(train_generator,steps_per_epoch=700,
                                 epochs=10,
                                 validation_data=validation_generator,
                                 validation_steps=990//batch_size)
```

이제 데이터 증가를 이용하기 위해 fit.generator를 사용하였다. 증가된 데이터의 개수는 앞의 배치사이즈 16와 step_per_epoch 700을 곱한 11200개이다. 실제 데이터의 약 3배되는 양으로 늘었으며, validation에 대해서는 증대를 시키지 않으므로 validation_steps는 validation의 총 개수를 16으로 나눈 몫으로 한다. 이 경우 결과는 Acc: 0.4501 Val_Acc: 0.4170이다. 상당히 정확도가 증가한 것을 알 수 있었다. 하지만 이 방법을 사용한 것도 정확도가 그렇게 높다고 할 수 없었다.

3. VGG 16을 사용한 모델

따라서 정확도를 더 높이기 위해 이전학습을 사용하여 더 많은 CNN 층으로 모델을 만들었다. 불러온 VGG16의 모수가 최신화되지 않도록 Trainable=False로 설정한다. 또한 이 VGG16을 사용하여 모델을 만드는 경우 VGG16 모델 뒤에 우리의 데이터에 맞게끔 MLP 층을 추가해 줄 수 있다. MLP층을 추가한 것으로 그치지 않고 많은 모수가 있기 때문에 과대적합이 발생하는 경우를 대비하여 MLP층에 L1, L2 규제화를 적용하였다. VGG16 모형에 MLP층을 추가한 모형은 다음과 같다.

```
vgg_base.trainable=False

additional_model = models.Sequential()
additional_model.add(vgg_base)
additional_model.add(Flatten())
additional_model.add(Dense(4096, activation='relu'))
additional_model.add(Dropout(0.5))
additional_model.add(Dense(1024, activation='relu',
                           kernel_regularizer=regularizers.l1_l2(l1=0.0001, l2=0.001)))
additional_model.add(Dropout(0.5))
additional_model.add(Dense(256, activation='relu',
                           kernel_regularizer=regularizers.l1_l2(l1=0.0001, l2=0.001)))
additional_model.add(Dropout(0.5))
additional_model.add(Dense(10, activation='softmax',
                           kernel_regularizer=regularizers.l1_l2(l1=0.0001, l2=0.001)))
additional_model.summary()
```

그림5. VGG16과 MLP3개층을 결합한 모형

이후 VGG16을 기반으로 하여 세가지 모형을 만들어 적합해 보았다. 그 결과 표는 아래와 같다.

index	Augment	Regularization	Trainable	Accuracy	Val_Accuracy
1	No	No	All False	0.9957	0.5848
2	Yes	Yes	All False	0.5832	0.5739
3	Yes	Yes	Final = True	0.7429	0.6191

그림 6. VGG16을 기반으로 한 모형의 정확도

첫째로 위 모형에 데이터의 증가 없이 적합해 보았다. 둘째로는 위 모형에 데이터를 추가하여 적합해 보았다. 마지막으로는 VGG16층의 마지막 CNN층의 모수도 최신화가 되게끔 진행하

여 더욱 정확도를 높여보려고 하였다. 이 작업은 이전에 VGG16 자체에 trainable=False를 적용한 것을 바꿔 마지막 CNN층에 대하여 최신화를 할 수 있게끔 해주어 진행하였다.

그림 6의 모델1은 VGG16에 MLP층만 추가한 경우 Accuracy가 VGG16을 추가하지 않은 모형에 비해 정확도가 증가한 것을 알 수 있다. 하지만 이 경우 과대적합이 심하게 되어있다. 따라서 과대적합을 막기 위해 L1, L2 정규화를 적용한 경우 두 정확도가 거의 비슷한 것을 보아 이는 과대적합이 많이 해결된 것을 알 수 있었다. 이후 VGG16의 마지막 층의 모수를 최신화 해줄 수 있게 해준 모델의 경우 정확도가 증가한 것을 알 수 있다. 하지만 과대적합이 다시 발생한 것을 알 수 있다. 이 데이터를 통해 인상주의 화가들을 분류하고자 하는 목적을 달성했다고 보기 어렵다. 이 부분은 'III 결론 및 한계점'에 자세히 서술하고자 한다.

분석2: 미술 사조가 다른 그림들을 분류해보자.

(1) 데이터 전처리

Art 데이터는 입체주의, 표현주의, 낭만주의 그림들로 구성되어 있다. 각각 Train 데이터로 사조 당 256개가 있으며 Test 데이터로는 사조 당 64개가 있다. 따라서 앞선 I 개요에서 설명했듯이 두 작가의 그림을 제외한 다른 작가의 그림에서 24개씩 뽑아 새로운 인상주의 파일을 만들어 옮겨주었다. 이후 분석1과 마찬가지로 ImageDataGenerator를 사용하여 데이터를 파이썬으로 불러오는 작업을 시도하였다.

따라서 앞선 설명과 같이 Train Data가 총 1024개로 되어 있으며, 4개의 범주로 나뉘어진 것을 알 수 있었다. Validation Data 또한 총 1024로 되어 있으며, 4개의 범주로 나뉘어졌다. 이후 분석을 하기 위해 각 데이터를 255로 나눠 RGB에 해당하는 각 픽셀의 값은 0과 1사이가 되게끔 조정해주었다.

(2) 모델 적용

1. 기본 모형

다른 모형들과 비교할 기본 모형을 만들기 위해 최대한 효율적인 작업을 위해 앞선 작업을 통해 알게 된 사실을 이용하였다. 기본 모형에서 가장 과대적합이 적은 모델은 출력층 직전의 MLP층을 줄이고 DropOut을 적용하였다. 앞선 모델들과 마찬가지로 분류할 범주가 4개이므로 활성함수로 Softmax를 사용하였고, 손실함수로는 Categorical_Crossentropy를 사용하였다. 이 경우 정확도는 Epoch 20에서 각각 Acc: 0.6416, Val_acc: 0.5508이다.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 256)       295168
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                401440
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 132
=================================================================
Total params: 789,988
Trainable params: 789,988
Non-trainable params: 0
_________________________________________________________________
```

```python
from keras.layers import Conv2D,MaxPooling2D,Flatten,Dense, Dropout
from keras import models
art_model=models.Sequential()
art_model.add(Conv2D(32,(3,3),activation='relu',input_shape=(150,150,3)))
art_model.add(MaxPooling2D(2,2))
art_model.add(Conv2D(64,(3,3),activation='relu'))
art_model.add(MaxPooling2D(2,2))
art_model.add(Conv2D(128,(3,3),activation='relu'))
art_model.add(MaxPooling2D(2,2))
art_model.add(Conv2D(256,(3,3),activation='relu'))
art_model.add(MaxPooling2D(2,2))
art_model.add(Dropout(0.3))
art_model.add(Flatten())
art_model.add(Dense(32,activation='relu'))
art_model.add(Dense(4,activation='softmax'))
art_model.summary()
```

그림7. 기본 모형

2. 데이터 증대 모델(Data Augment)

데이터 증대를 활용한 모델을 만들어 데이터에 적합해보고자 하였다. 데이터를 증가시키 위해 기존 데이터에 변형을 가할 때, 앞과 마찬가지의 이유로 fill_mode를 지정할 때, 데이터 특성을 고려해야 했다. 따라서 앞선 분석에서 reflect를 사용하였고, 이번에는 Wrap을 사용해 보았다. 두 옵션의 차이는 단지 데이터가 거울에 맺힌 상처럼 반대로 계속 이어지느냐의 여부이다.

또한 다른 변형 정도도 앞선 분석과 같게 조정해 주었다. 이후, 배치 사이즈를 16으로 조정해 주었다.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 15, 15, 256)       295168
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 5, 5, 512)         1180160
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                65568
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 132
=================================================================
Total params: 1,634,276
Trainable params: 1,634,276
Non-trainable params: 0
_________________________________________________________________
```

그림8. 데이터 증대(Augment)를 사용한 모형

```python
import instancenormalization
inputs = Input(shape=(150, 150, 3))
x = Conv2D(32, (3, 3))(inputs)
x = instancenormalization.InstanceNormalization(
    axis=-1,center=False, scale=False)(x)
x = Activation('relu')(x)
x =  MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x =  MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(128, (3, 3), activation='relu')(x)
x =  MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(256, (3, 3), activation='relu')(x)
x =  MaxPooling2D((2, 2), padding='same')(x)
x = Flatten()(x)
x = Dropout(0.6)(x)
x = Dense(32,activation='relu')(x)
x = Dense(4,activation='softmax')(x)
aug_model = Model(inputs, x)
aug_model.summary()
```

데이터 증가를 사용한 모형을 만들기 위해 위 모델에 대하여 steps_per_epoch로 350을 지정해 총 데이터의 개수를 16*350 = 5600개로 늘려 분석을 실시하였다. 그 결과 정확도는 각각 (acc: 0.6408, Val_acc: 0.5787)이었다.

3. VGG16을 활용한 모델

VGG 16을 활용하여 모델의 깊이를 늘리고 MLP층을 추가하였다.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_3 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_5 (Dense)              (None, 2048)              16779264
_________________________________________________________________
dropout_3 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_6 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_7 (Dense)              (None, 4)                 4100
=================================================================
Total params: 33,596,228
Trainable params: 18,881,540
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
additional_model = models.Sequential()
additional_model.add(vgg_base)
additional_model.add(Flatten())
additional_model.add(Dense(2048, activation='relu',
                           kernel_regularizer=
                           regularizers.l1_l2(l1=0.0001, l2=0.001)))
additional_model.add(Dropout(0.5))
additional_model.add(Dense(1024, activation='relu',
                           kernel_regularizer=
                           regularizers.l1_l2(l1=0.0001, l2=0.001)))
additional_model.add(Dropout(0.5))
additional_model.add(Dense(4, activation='softmax',
                           kernel_regularizer=
                           regularizers.l1_l2(l1=0.0001, l2=0.001)))
additional_model.summary()
```

그림9 VGG16에 MLP 층을 추가한 모형

이전과 마찬가지로 VGG16 + MLP의 모형으로 데이터를 분석해보고, 데이터 증대를 적용하여 모형을 분석해 보았다. 두 모형 모두 마지막 출력층에 대한 정규화를 적용하였다. 그 결과는 다음과 같다.

	Augment	Regularization	Acc	Val_Acc
1	No	Yes	0.7451	0.7109
2	Yes	Yes	0.7208	0.7070

그림 10 VGG16을 기반으로 한 모형의 분류정확도

결과에 따르면 Augment의 효과에 의해 정확도가 증가한 것을 알 수 있었다. 다만 정확도가 증가하면서 과대적합이 발생하였다.

III. 결론 및 한계점

Impression Data에 대한 분석결과는 다음과 같다.

index	모델	Acc	Val_Acc
1	기본 모델	0.4132	0.3576
2	Augment	0.4501	0.4170
3	VGG + AUG	0.5832	0.5739

Art Data에 대한 분석결과는 다음과 같다.

index	모델	Acc	Val_Acc
1	기본 모델	0.6416	0.5508
2	Augment	0.6408	0.5787
3	VGG + AUG	0.7208	0.7070

먼저, 각 데이터에서 공통적으로 나타나는 특징을 서술하고자 한다. 기본 모델의 경우 과대적합이 발생하면 출력층 직전의 노드 수를 줄이고 Dropout을 쓰거나 L1, L2 규제를 통해 두 정확도를 맞춰주는 것이 효과적이다. 하지만 정확도면에 봤을 땐, 이 작업이 정확도 자체를 올려주지 못한다.

따라서 정확도를 올리기 위해 가장 좋은 것은 데이터 수를 증가시키는 것이다. 다만, 다른 실제 상황에서도 유사하듯이 다른 데이터를 구하는 것은 어렵다. 분석한 데이터를 예로 들면, 데이터를 찾거나 만들 재화가 충분한 경우에도 한 화가의 작품 수가 많지 않아 더 이상 찾을 수 없는 경우엔 데이터를 늘릴 수 없다. 하지만 Augment 방법을 통해 데이터(그림)를 최대한 특성을 유지시킨 채 변형시켜 모델 적합에 사용한다면 상당한 효과를 확인할 수 있다.

이후 이전학습 모형인 VGG16을 사용하여 이미 많은 데이터에 적합된 모델을 이전 학습하여 Augment와 같이 사용하면 기본 모형에 비해 정확도가 올라가고 과대적합 문제를 해결하는 목적에 대해 상당한 효과를 볼 수 있다.

다만 위 두가지 표에 나타난 대로 과대적합 해결이 어렵고 분류의 정확도가 높지 못했다. 그 이유는 두 가지로 볼 수 있다. 데이터의 수가 범주에 비해 높지 않았다. 즉, 데이터의 수가 많지 않아서 기본적인 분류가 어려웠다. 이는 Augment로 어느정도 해결해 줄 수 있었다. 하지만 그 정확도가 그래도 높지 못했는데, 그 이유는 바로 데이터가 가진 특성 때문이다. 이 데이터를 가지고 분류하고자 했으나 인상주의 화가만 포함된 데이터의 경우 같은 미술 사조에 포함된 그림의 경우 비슷한 색채 사용의 양상을 갖기 때문에 화가에 대한 분류는 어려웠다. 더불어 Art 데이터는 다른 미술 사조라는 점과 분류의 가짓수가 적다는 점에서 Impression 데이터에 비해 예측율이 높았으나 색채를 강조한 미술 사조는 인상주의만이라는 것 때문에 예측율이 눈에 띄게 높지는 못했다.

하지만, 신경망 구조가 가진 잠재적인 특성을 찾아낼 수 있다는 점에 기대하고 분류를 진행해본 결과 월등하게 예측 정확도가 높다고 할 수는 없으나 여러가지 기법들을 통해 예측율을 높일 수 있었고 최종 모델의 정확도인 0.5739, 0.7070은 각각 범주가 10개와 4개인 것을 고려하면 낮은 수치라고 할 수 없다.